Bit Vector Data Flow Frameworks

Uday Khedker
(www.cse.iitb.ac.in/~uday)

Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

Jul 2017
Part 1

About These Slides
Copyright

These slides constitute the lecture notes for CS618 Program Analysis course at IIT Bombay and have been made available as teaching material accompanying the book:

  (Indian edition published by Ane Books in 2013)

Apart from the above book, some slides are based on the material from the following books


*These slides are being made available under GNU FDL v1.2 or later purely for academic or research use.*
Outline

- Live Variables Analysis
- Observations about Data Flow Analysis
- Available Expressions Analysis
- Anticipable Expressions Analysis
- Reaching Definitions Analysis
- Common Features of Bit Vector Frameworks
- Partial Redundancy Elimination
Part 2

Live Variables Analysis
Defining Live Variables Analysis

A variable \( v \) is live at a program point \( p \), if some path from \( p \) to program exit contains an r-value occurrence of \( v \) which is not preceded by an l-value occurrence of \( v \).

\[
\begin{align*}
  v &= a \times b \\
  a &= v + 2 \\
  v &= a + 2
\end{align*}
\]
Defining Live Variables Analysis

A variable $v$ is live at a program point $p$, if some path from $p$ to program exit contains an r-value occurrence of $v$ which is not preceded by an l-value occurrence of $v$. 

$v$ is live at $p$
A variable $v$ is live at a program point $p$, if some path from $p$ to program exit contains an r-value occurrence of $v$ which is not preceded by an l-value occurrence of $v$. 

$v$ is live at $p$

$v$ is not live at $p$
Defining Live Variables Analysis

A variable $v$ is live at a program point $p$, if some path from $p$ to program exit contains an r-value occurrence of $v$ which is not preceded by an l-value occurrence of $v$. 

$v$ is live at $p$

$v$ is not live at $p$

$v$ is live at $p$
Defining Live Variables Analysis

A variable $v$ is live at a program point $p$, if some path from $p$ to program exit contains an r-value occurrence of $v$ which is not preceded by an l-value occurrence of $v$.

$v$ is live at $p$

$v$ is not live at $p$

$v$ is live at $p$

Path based specification
Defining Data Flow Analysis for Live Variables Analysis

\[
\begin{align*}
\text{In}_i & = \text{Gen}_i \
& \quad \cup \left( \text{Out}_i - \text{Kill}_i \right) \\
\text{Gen}_k & = \text{In}_i \cup \text{In}_j \\
\text{Out}_k & = \text{In}_i \\
\end{align*}
\]
Defining Data Flow Analysis for Live Variables Analysis

Basic Blocks $\equiv$ Single statements or Maximal groups of sequentially executed statements
Defining Data Flow Analysis for Live Variables Analysis

Basic Blocks $\equiv$ Single statements or Maximal groups of sequentially executed statements

Control Transfer
Defining Data Flow Analysis for Live Variables Analysis

\[ \text{Gen}_i, \text{Kill}_i \]

\[ \text{Gen}_j, \text{Kill}_j \]

\[ \text{Gen}_k, \text{Kill}_k \]
Defining Data Flow Analysis for Live Variables Analysis

\[
\begin{align*}
\text{Gen}_k, \text{Kill}_k & \\
\text{Gen}_i, \text{Kill}_i & \\
\text{Gen}_j, \text{Kill}_j & \\
\text{Local Data Flow Properties} & \\
\end{align*}
\]
Local Data Flow Properties for Live Variables Analysis

\[ Gen_n = \{ v \mid \text{variable } v \text{ is used in basic block } n \text{ and is not preceded by a definition of } v \} \]

\[ Kill_n = \{ v \mid \text{basic block } n \text{ contains a definition of } v \} \]
Local Data Flow Properties for Live Variables Analysis

**r-value occurrence**
Value is only read, e.g. \(x, y, z\) in
\[
x.\text{sum} = y.\text{data} + z.\text{data}
\]

\[
\text{Gen}_n = \{ \nu \mid \text{variable } \nu \text{ is used in basic block } n \text{ and is not preceded by a definition of } \nu \}
\]

\[
\text{Kill}_n = \{ \nu \mid \text{basic block } n \text{ contains a definition of } \nu \}
\]
Local Data Flow Properties for Live Variables Analysis

**r-value occurrence**
Value is only read, e.g. \(x, y, z\) in
\[
x.\text{sum} = y.\text{data} + z.\text{data}
\]

**l-value occurrence**
Value is modified e.g. \(y\) in
\[
y = x.\text{lptr}
\]

\[
\text{Gen}_n = \{ \nu | \text{variable } \nu \text{ is used in basic block } n \text{ and is not preceded by a definition of } \nu \} 
\]

\[
\text{Kill}_n = \{ \nu | \text{basic block } n \text{ contains a definition of } \nu \}
\]
Local Data Flow Properties for Live Variables Analysis

**r-value occurrence**
Value is only read, e.g.

\[ x, y, z \text{ in } x.\text{sum} = y.\text{data} + z.\text{data} \]

**l-value occurrence**
Value is modified, e.g.

\[ y = x.\text{lptr} \]

**Gen**

\[ \text{Gen}_n = \{ \text{v} \mid \text{variable v is used in basic block n and is not preceded by a definition of v} \} \]

**Kill**

\[ \text{Kill}_n = \{ \text{v} \mid \text{basic block n contains a definition of v} \} \]
Local Data Flow Properties for Live Variables Analysis

\[ Gen_n = \{ v \mid \text{variable } v \text{ is used in basic block } n \text{ and is not preceded by a definition of } v \} \]

\[ Kill_n = \{ v \mid \text{basic block } n \text{ contains a definition of } v \} \]

r-value occurrence
Value is only read, e.g. \( x, y, z \) in
\[ x.\text{sum} = y.\text{data} + z.\text{data} \]

l-value occurrence
Value is modified e.g. \( y \) in
\[ y = x.\text{lptr} \]

within \( n \)

anywhere in \( n \)
Defining Data Flow Analysis for Live Variables Analysis

\[ \text{In}_k = \text{Gen}_k \cup (\text{Out}_k - \text{Kill}_k) \]

\[ \text{Out}_k = \text{In}_i \cup \text{In}_j \]

\[ \text{In}_i \]

\[ \text{In}_j \]
Defining Data Flow Analysis for Live Variables Analysis

Global Data Flow Properties

\[ \text{\text{In}}_k = \text{Gen}_k \cup (\text{Out}_k - \text{Kill}_k) \]

\[ \text{Out}_k = \text{In}_i \cup \text{In}_j \]

\[ \text{In}_i \]

\[ \text{In}_j \]
Defining Data Flow Analysis for Live Variables Analysis

Global Data Flow Properties

\[ \text{In}_k = \text{Gen}_k \cup (\text{Out}_k - \text{Kill}_k) \]

\[ \text{Out}_k = \text{In}_i \cup \text{In}_j \]

Edge based specifications
Data Flow Equations For Live Variables Analysis

\[ In_n = (Out_n - Kill_n) \cup Gen_n \]

\[ Out_n = \begin{cases} 
    BI & \text{if } n \text{ is } \text{End} \text{ block} \\
    \bigcup_{s \in succ(n)} In_s & \text{otherwise} 
\end{cases} \]
Data Flow Equations For Live Variables Analysis

\[ \text{In}_n = (\text{Out}_n - \text{Kill}_n) \cup \text{Gen}_n \]

\[ \text{Out}_n = \begin{cases} \text{Bl} & \text{n is End block} \\ \bigcup_{s \in \text{succ}(n)} \text{In}_s & \text{otherwise} \end{cases} \]

- \( \text{In}_n \) and \( \text{Out}_n \) are sets of variables
Data Flow Equations For Live Variables Analysis

\[ In_n = (Out_n - Kill_n) \cup Gen_n \]

\[ Out_n = \begin{cases} \bigcup_{s \in succ(n)} In_s & \text{otherwise} \\ BI & \text{n is End block} \end{cases} \]

- \( In_n \) and \( Out_n \) are sets of variables
- \( BI \) is boundary information representing the effect of calling contexts
  - \( \emptyset \) for local variables except for the values being returned
  - set of global variables used further in any calling context
    (can be safely approximated by the set of all global variables)
Data Flow Equations for Our Example

\[ w = x \]
\[ \text{while (x.data < max)} \]
\[ y = x.lptr \]
\[ x = x.rptr \]
\[ z = \text{New class of } z \]
\[ y = y.lptr \]
\[ z.\text{sum} = x.\text{data} + y.\text{data} \]

\[ \text{In}_1 = (\text{Out}_1 - \text{Kill}_1) \cup \text{Gen}_1 \]
\[ \text{Out}_1 = \text{In}_2 \]
\[ \text{In}_2 = (\text{Out}_2 - \text{Kill}_2) \cup \text{Gen}_2 \]
\[ \text{Out}_2 = \text{In}_3 \cup \text{In}_4 \]
\[ \text{In}_3 = (\text{Out}_3 - \text{Kill}_3) \cup \text{Gen}_3 \]
\[ \text{Out}_3 = \text{In}_2 \]
\[ \text{In}_4 = (\text{Out}_4 - \text{Kill}_4) \cup \text{Gen}_4 \]
\[ \text{Out}_4 = \text{In}_5 \]
\[ \text{In}_5 = (\text{Out}_5 - \text{Kill}_5) \cup \text{Gen}_5 \]
\[ \text{Out}_5 = \text{In}_6 \]
\[ \text{In}_6 = (\text{Out}_6 - \text{Kill}_6) \cup \text{Gen}_6 \]
\[ \text{Out}_6 = \text{In}_7 \]
\[ \text{In}_7 = (\text{Out}_7 - \text{Kill}_7) \cup \text{Gen}_7 \]
\[ \text{Out}_7 = \emptyset \]
Data Flow Equations for Our Example

1. \[ w \leftarrow x \]
2. \[
\text{while (} x\text{.data} < \text{max})
\]
3. \[ x \leftarrow x\text{.rptr} \]
4. \[ y \leftarrow x\text{.lptr} \]
5. \[ z \leftarrow \text{New class of } z \]
6. \[ y \leftarrow y\text{.lptr} \]
7. \[ z\text{.sum} \leftarrow x\text{.data} + y\text{.data} \]

\[
\begin{align*}
\text{In}_1 &= (\text{Out}_1 - \text{Kill}_1) \cup \text{Gen}_1 \\
\text{Out}_1 &= \text{In}_2 \\
\text{In}_2 &= (\text{Out}_2 - \text{Kill}_2) \cup \text{Gen}_2 \\
\text{Out}_2 &= \text{In}_3 \cup \text{In}_4 \\
\text{In}_3 &= (\text{Out}_3 - \text{Kill}_3) \cup \text{Gen}_3 \\
\text{Out}_3 &= \text{In}_2 \\
\text{In}_4 &= (\text{Out}_4 - \text{Kill}_4) \cup \text{Gen}_4 \\
\text{Out}_4 &= \text{In}_5 \\
\text{In}_5 &= (\text{Out}_5 - \text{Kill}_5) \cup \text{Gen}_5 \\
\text{Out}_5 &= \text{In}_6 \\
\text{In}_6 &= (\text{Out}_6 - \text{Kill}_6) \cup \text{Gen}_6 \\
\text{Out}_6 &= \text{In}_7 \\
\text{In}_7 &= (\text{Out}_7 - \text{Kill}_7) \cup \text{Gen}_7 \\
\text{Out}_7 &= \emptyset
\end{align*}
\]
Data Flow Equations for Our Example

\[\begin{align*}
In_1 &= (Out_1 - Kill_1) \cup Gen_1 \\
Out_1 &= In_2 \\
In_2 &= (Out_2 - Kill_2) \cup Gen_2 \\
Out_2 &= In_3 \cup In_4 \\
In_3 &= (Out_3 - Kill_3) \cup Gen_3 \\
Out_3 &= In_2 \\
In_4 &= (Out_4 - Kill_4) \cup Gen_4 \\
Out_4 &= In_5 \\
In_5 &= (Out_5 - Kill_5) \cup Gen_5 \\
Out_5 &= In_6 \\
In_6 &= (Out_6 - Kill_6) \cup Gen_6 \\
Out_6 &= In_7 \\
In_7 &= (Out_7 - Kill_7) \cup Gen_7 \\
Out_7 &= \emptyset
\end{align*}\]
Data Flow Equations for Our Example

\[ w \equiv x \]

\[ \text{while (} x.\text{data} < \text{max}) \]

\[ y \equiv x.\text{rptr} \]

\[ z \equiv \text{New class of} \ z \]

\[ y \equiv y.\text{lptr} \]

\[ \text{z.sum} = x.\text{data} + y.\text{data} \]

\[ \text{In} 1 = (\text{Out} 1 - \text{Kill} 1) \cup \text{Gen} 1 \]

\[ \text{Out} 1 = \text{In} 2 \]

\[ \text{In} 2 = (\text{Out} 2 - \text{Kill} 2) \cup \text{Gen} 2 \]

\[ \text{Out} 2 = \text{In} 3 \cup \text{In} 4 \]

\[ \text{In} 3 = (\text{Out} 3 - \text{Kill} 3) \cup \text{Gen} 3 \]

\[ \text{Out} 3 = \text{In} 2 \]

\[ \text{In} 4 = (\text{Out} 4 - \text{Kill} 4) \cup \text{Gen} 4 \]

\[ \text{Out} 4 = \text{In} 5 \]

\[ \text{In} 5 = (\text{Out} 5 - \text{Kill} 5) \cup \text{Gen} 5 \]

\[ \text{Out} 5 = \text{In} 6 \]

\[ \text{In} 6 = (\text{Out} 6 - \text{Kill} 6) \cup \text{Gen} 6 \]

\[ \text{Out} 6 = \text{In} 7 \]

\[ \text{In} 7 = (\text{Out} 7 - \text{Kill} 7) \cup \text{Gen} 7 \]

\[ \text{Out} 7 = \emptyset \]
Data Flow Equations for Our Example

1. \( w = x \)

2. while \((x.data < \text{max})\)

3. \( x = x.rptr \)

4. \( y = x.lptr \)

5. \( z = \text{New class of } z \)

6. \( y = y.lptr \)

7. \( z.sum = x.data + y.data \)

\[ \begin{align*}
    \text{Out}_1 &= \text{In}_2 \\
    \text{In}_1 &= (\text{Out}_1 - \text{Kill}_1) \cup \text{Gen}_1 \\
    \text{Out}_2 &= \text{In}_3 \cup \text{In}_4 \\
    \text{In}_2 &= (\text{Out}_2 - \text{Kill}_2) \cup \text{Gen}_2 \\
    \text{Out}_3 &= \text{In}_2 \\
    \text{In}_3 &= (\text{Out}_3 - \text{Kill}_3) \cup \text{Gen}_3 \\
    \text{Out}_4 &= \text{In}_5 \\
    \text{In}_4 &= (\text{Out}_4 - \text{Kill}_4) \cup \text{Gen}_4 \\
    \text{Out}_5 &= \text{In}_6 \\
    \text{In}_5 &= (\text{Out}_5 - \text{Kill}_5) \cup \text{Gen}_5 \\
    \text{Out}_6 &= \text{In}_7 \\
    \text{In}_6 &= (\text{Out}_6 - \text{Kill}_6) \cup \text{Gen}_6 \\
    \text{Out}_7 &= \emptyset \\
    \text{In}_7 &= (\text{Out}_7 - \text{Kill}_7) \cup \text{Gen}_7
\end{align*} \]
Data Flow Equations for Our Example

\[ \begin{align*}
In_1 &= (Out_1 - Kill_1) \cup Gen_1 \\
Out_1 &= In_2 \\
In_2 &= (Out_2 - Kill_2) \cup Gen_2 \\
Out_2 &= In_3 \cup In_4 \\
In_3 &= (Out_3 - Kill_3) \cup Gen_3 \\
Out_3 &= In_2 \\
In_4 &= (Out_4 - Kill_4) \cup Gen_4 \\
Out_4 &= In_5 \\
In_5 &= (Out_5 - Kill_5) \cup Gen_5 \\
Out_5 &= In_6 \\
In_6 &= (Out_6 - Kill_6) \cup Gen_6 \\
Out_6 &= In_7 \\
In_7 &= (Out_7 - Kill_7) \cup Gen_7 \\
Out_7 &= \emptyset
\end{align*} \]
Data Flow Equations for Our Example

1. \( w \equiv x \)
2. \( \text{while } (x.\text{data} < \text{max}) \)
3. \( x = x.\text{rptr} \)
4. \( y = x.\text{lptr} \)
5. \( z = \text{New class of } z \)
6. \( y = y.\text{lptr} \)
7. \( z.\text{sum} = x.\text{data} + y.\text{data} \)

\[
\begin{align*}
\text{In}_1 &= (\text{Out}_1 - \text{Kill}_1) \cup \text{Gen}_1 \\
\text{Out}_1 &= \text{In}_2 \\
\text{In}_2 &= (\text{Out}_2 - \text{Kill}_2) \cup \text{Gen}_2 \\
\text{Out}_2 &= \text{In}_3 \cup \text{In}_4 \\
\text{In}_3 &= (\text{Out}_3 - \text{Kill}_3) \cup \text{Gen}_3 \\
\text{Out}_3 &= \text{In}_2 \\
\text{In}_4 &= (\text{Out}_4 - \text{Kill}_4) \cup \text{Gen}_4 \\
\text{Out}_4 &= \text{In}_5 \\
\text{In}_5 &= (\text{Out}_5 - \text{Kill}_5) \cup \text{Gen}_5 \\
\text{Out}_5 &= \text{In}_6 \\
\text{In}_6 &= (\text{Out}_6 - \text{Kill}_6) \cup \text{Gen}_6 \\
\text{Out}_6 &= \text{In}_7 \\
\text{In}_7 &= (\text{Out}_7 - \text{Kill}_7) \cup \text{Gen}_7 \\
\text{Out}_7 &= \emptyset
\end{align*}
\]
Performing Live Variables Analysis

Gen = {x}, Kill = {w}
\[ w = x \]

Gen = {x}, Kill = ∅
while (x.data < max)

Gen = {x}, Kill = {y}
y = x.lptr

Gen = ∅, Kill = {z}
z = New class of z

Gen = {y}, Kill = {y}
y = y.lptr

Gen = {x, y, z}, Kill = ∅
z.sum = x.data + y.data
Performing Live Variables Analysis

Gen = \{x\}, \text{Kill} = \{w\}
\quad w = x

Gen = \{x\}, \text{Kill} = \emptyset
\quad \text{while} (x.\text{data} < \text{max})

Gen = \{x\}, \text{Kill} = \{y\}
\quad y = x.lptr

Gen = \emptyset, \text{Kill} = \{z\}
\quad z = \text{New class of } z

Gen = \{y\}, \text{Kill} = \{y\}
\quad y = y.lptr

Gen = \{x, y, z\}, \text{Kill} = \emptyset
\quad z.\text{sum} = x.\text{data} + y.\text{data}

Gen and Kill need not be mutually exclusive
Performing Live Variables Analysis

\[ \text{Gen} = \{x\}, \text{Kill} = \{w\} \]
\[ w = x \]

\[ \text{Gen} = \{x\}, \text{Kill} = \emptyset \]
while \((x.\text{data} < \text{max})\)

\[ \text{Gen} = \{x\}, \text{Kill} = \{y\} \]
\[ y = x.\text{lptr} \]

\[ \text{Gen} = \emptyset, \text{Kill} = \{z\} \]
\[ z = \text{New \ class\ of\ z} \]

\[ \text{Gen} = \{y\}, \text{Kill} = \{y\} \]
\[ y = y.\text{lptr} \]

\[ \text{Gen} = \{x, y, z\}, \text{Kill} = \emptyset \]
\[ z.\text{sum} = x.\text{data} + y.\text{data} \]

\[ z \text{ is an r-value occurrence and not an l-value occurrence} \]
Performing Live Variables Analysis

Gen = \{ x \}, \text{Kill} = \{ w \}
\hspace{0.5cm} w = x

Gen = \{ x \}, \text{Kill} = \emptyset
while (x.data < max)

Gen = \{ x \}, \text{Kill} = \{ y \}
\hspace{0.5cm} y = x.\text{lptr}

Gen = \emptyset, \text{Kill} = \{ z \}
\hspace{0.5cm} z = \text{New class of z}

Gen = \{ y \}, \text{Kill} = \{ y \}
\hspace{0.5cm} y = y.\text{lptr}

Gen = \{ x, y, z \}, \text{Kill} = \emptyset
\hspace{0.5cm} z.\text{sum} = x.\text{data} + y.\text{data}

x, y, z are considered to be used based purely on local use even if the value of z is not used later. A different analysis called strongly live variables analysis improves on this.
Performing Live Variables Analysis

Initialization

Gen = {x}, Kill = {w}
\[ w = x \]

Gen = {x}, Kill = ∅
\[ \text{while } (x \text{.data } < \text{max}) \]

Gen = {x}, Kill = {y}
\[ y = x \text{.lptr} \]

Gen = {x}, Kill = {x}
\[ x = x \text{.rptr} \]

Gen = {x}, Kill = {y}
\[ y = y \text{.lptr} \]

Gen = {x, y, z}, Kill = ∅
\[ z \text{.sum } = x \text{.data } + y \text{.data} \]
Performing Live Variables Analysis

Ignoring max because we are doing analysis for pointer variables w, x, y, z

Gen = \{x\}, Kill = \{w\}

{w = x}

Gen = \{x\}, Kill = \{\emptyset\}

while (x.data < max)

Gen = \{x\}, Kill = \{\emptyset\}

y = x.rptr

Gen = \{x\}, Kill = \{\emptyset\}

y = y.lptr

Gen = \{x, y, z\}, Kill = \{\emptyset\}

z = New class of z

z.sum = x.data + y.data

Iteration #1

Traversal
Performing Live Variables Analysis

Ignoring max because we are doing analysis for pointer variables w, x, y, z

\[
\begin{align*}
Gen &= \{x\}, \quad Kill = \{w\} \\
\{x\} &\quad \{x\} \\
\{x\} &\quad \{x\} \\
\{x\} &\quad \{x\}
\end{align*}
\]

\[
\begin{align*}
w &= x \\
Gen &= \{x\}, \quad Kill = \emptyset \\
\{x\} &\quad \{x\} \\
\{x\} &\quad \{x\}
\end{align*}
\]

\[
\begin{align*}
while (x.data < max) \\
Gen &= \{x\}, \quad Kill = \emptyset \\
\{x\} &\quad \{x\} \\
\{x\} &\quad \{x\}
\end{align*}
\]

\[
\begin{align*}
x &= x.rptr \\
Gen &= \{x\}, \quad Kill = \{x\} \\
\{x\} &\quad \{x\} \\
\{x\} &\quad \{x\}
\end{align*}
\]

\[
\begin{align*}
y &= x.lptr \\
Gen &= \emptyset, \quad Kill = \{z\} \\
\{x, y\} &\quad \{x, y\} \\
\{x, y\} &\quad \{x, y\}
\end{align*}
\]

\[
\begin{align*}
z &= \text{New class of } z \\
Gen &= \emptyset, \quad Kill = \{z\} \\
\{x, y, z\} &\quad \{x, y, z\} \\
\{x, y, z\} &\quad \{x, y, z\}
\end{align*}
\]

\[
\begin{align*}
y &= y.lptr \\
Gen &= \{y\}, \quad Kill = \{y\} \\
\{x, y, z\} &\quad \{x, y, z\} \\
\{x, y, z\} &\quad \{x, y, z\}
\end{align*}
\]

\[
\begin{align*}
z &= \text{New class of } z \\
Gen &= \emptyset, \quad Kill = \{z\} \\
\{x, y, z\} &\quad \{x, y, z\} \\
\{x, y, z\} &\quad \{x, y, z\}
\end{align*}
\]

\[
\begin{align*}
z &= \text{New class of } z \\
Gen &= \{x, y, z\}, \quad Kill = \emptyset \\
z\.sum &= x.data + y.data \\
\{x, y, z\} &\quad \{x, y, z\} \\
\{x, y, z\} &\quad \{x, y, z\}
\end{align*}
\]
Performing Live Variables Analysis

Local data flow properties when basic blocks contain multiple statements

\[
\text{Gen} = \{x\}, \quad \text{Kill} = \{w\} \\
w = x
\]

\[
\text{Gen} = \{x\}, \quad \text{Kill} = \emptyset \\
\text{while} \ (x.\text{data} < \text{max})
\]

\[
\text{Gen} = \{x\}, \quad \text{Kill} = \{y, z\} \\
y = x.\text{lptr} \\
z = \text{New class of } z \\
y = y.\text{lptr} \\
z.\text{sum} = x.\text{data} + y.\text{data}
\]

\[
\text{Gen} = \{x\}, \quad \text{Kill} = \{x\} \\
x = x.\text{rptr}
\]
Local Data Flow Properties for Live Variables Analysis

\[ In_n = Gen_n \cup (Out_n - Kill_n) \]

- \( Gen_n \) : Use not preceded by definition
- \( Kill_n \) : Definition anywhere in a block
Local Data Flow Properties for Live Variables Analysis

\[ \text{In}_n = \text{Gen}_n \cup (\text{Out}_n - \text{Kill}_n) \]

- **Gen}_n : Use not preceded by definition
  
  *Upwards exposed use*

- **Kill}_n : Definition anywhere in a block
  
  *Stop the effect from being propagated across a block*
## Local Data Flow Properties for Live Variables Analysis

<table>
<thead>
<tr>
<th>Case</th>
<th>Local Information</th>
<th>Example</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$v \notin Gen_n$</td>
<td>$v \notin Kill_n$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$v \in Gen_n$</td>
<td>$v \notin Kill_n$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$v \notin Gen_n$</td>
<td>$v \in Kill_n$</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$v \in Gen_n$</td>
<td>$v \in Kill_n$</td>
<td></td>
</tr>
</tbody>
</table>
## Local Data Flow Properties for Live Variables Analysis

<table>
<thead>
<tr>
<th>Case</th>
<th>Local Information</th>
<th>Example</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( v \notin Gen_n ) ( v \notin Kill_n )</td>
<td>( a = b + c ) ( b = c \times d )</td>
<td>liveness of ( v ) is unaffected by the basic block</td>
</tr>
<tr>
<td>2</td>
<td>( v \in Gen_n ) ( v \notin Kill_n )</td>
<td>( a = b + c ) ( b = v \times d )</td>
<td>( v ) becomes live before the basic block</td>
</tr>
<tr>
<td>3</td>
<td>( v \notin Gen_n ) ( v \in Kill_n )</td>
<td>( a = b + c ) ( v = c \times d )</td>
<td>( v ) ceases to be live before the basic block</td>
</tr>
<tr>
<td>4</td>
<td>( v \in Gen_n ) ( v \in Kill_n )</td>
<td>( a = v + c ) ( v = c \times d )</td>
<td>liveness of ( v ) is killed but ( v ) becomes live before the basic block</td>
</tr>
</tbody>
</table>
Using Data Flow Information of Live Variables Analysis

- Used for register allocation

  If variable $x$ is live in a basic block $b$, it is a potential candidate for register allocation
Using Data Flow Information of Live Variables Analysis

- Used for register allocation
  If variable $x$ is live in a basic block $b$, it is a potential candidate for register allocation

- Used for dead code elimination
  If variable $x$ is not live after an assignment $x = \ldots$, then the assignment is redundant and can be deleted as dead code
**Tutorial Problem 1: Perform Dead Code Elimination**

```
a = 4
b = 2
c = 3
n = c*2

if (a > n)
    a = a+1

if (a ≥ 12)
    t1 = a+b
    a = t1+c
    print "Hi"

print "Hello"
```

**Local Data Flow Information**

<table>
<thead>
<tr>
<th></th>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1</td>
<td>∅</td>
<td>{a, b, c, n}</td>
</tr>
<tr>
<td>n2</td>
<td>{a, n}</td>
<td>∅</td>
</tr>
<tr>
<td>n3</td>
<td>{a}</td>
<td>{a}</td>
</tr>
<tr>
<td>n4</td>
<td>{a}</td>
<td>∅</td>
</tr>
<tr>
<td>n5</td>
<td>{a, b, c}</td>
<td>{a, t1}</td>
</tr>
<tr>
<td>n6</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>
Tutorial Problem 1: Perform Dead Code Elimination

Local Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1</td>
<td>{a, b, c, n}</td>
<td>{}</td>
</tr>
<tr>
<td>n2</td>
<td>{a, n}</td>
<td>{}</td>
</tr>
<tr>
<td>n3</td>
<td>{a}</td>
<td>{a}</td>
</tr>
<tr>
<td>n4</td>
<td>{a}</td>
<td>{}</td>
</tr>
<tr>
<td>n5</td>
<td>{a, b, c}</td>
<td>{a, t1}</td>
</tr>
<tr>
<td>n6</td>
<td>{}</td>
<td>{}</td>
</tr>
</tbody>
</table>

Global Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Iteration #1</th>
<th>Iteration #2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>n6</td>
<td>{}</td>
<td></td>
</tr>
<tr>
<td>n5</td>
<td>{}</td>
<td>{a, b, c}</td>
</tr>
<tr>
<td>n4</td>
<td>{a, b, c}</td>
<td>{a, b, c}</td>
</tr>
<tr>
<td>n3</td>
<td>{}</td>
<td>{a}</td>
</tr>
<tr>
<td>n2</td>
<td>{a, b, c}</td>
<td>{a, b, c, n}</td>
</tr>
<tr>
<td>n1</td>
<td>{a, b, c, n}</td>
<td>{}</td>
</tr>
</tbody>
</table>
**Tutorial Problem 1: Perform Dead Code Elimination**

```plaintext
a = 4
b = 2
c = 3
n = c*2

if (a > n)
    n2
T
n3
F
a = a+1
n4
if (a ≥ 12)
    n5
T
f1 = a+b
a = f1+c
n5
print "Hi"
print "Hello"
n6
```

### Local Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1</td>
<td>∅</td>
<td>{a, b, c, n}</td>
</tr>
<tr>
<td>n2</td>
<td>{a, n}</td>
<td>∅</td>
</tr>
<tr>
<td>n3</td>
<td>{a}</td>
<td>{a}</td>
</tr>
<tr>
<td>n4</td>
<td>{a}</td>
<td>∅</td>
</tr>
<tr>
<td>n5</td>
<td>{a, b, c}</td>
<td>{a, t1}</td>
</tr>
<tr>
<td>n6</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>

### Global Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Iteration #1</th>
<th>Iteration #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>In</td>
<td>Out</td>
</tr>
<tr>
<td>n6</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>n5</td>
<td>∅</td>
<td>{a, b, c}</td>
</tr>
<tr>
<td>n4</td>
<td>{a, b, c}</td>
<td>{a, b, c}</td>
</tr>
<tr>
<td>n3</td>
<td>∅</td>
<td>{a}</td>
</tr>
<tr>
<td>n2</td>
<td>{a, b, c}</td>
<td>{a, b, c, n}</td>
</tr>
<tr>
<td>n1</td>
<td>{a, b, c, n}</td>
<td>∅</td>
</tr>
</tbody>
</table>
```
Tutorial Problem 1: Perform Dead Code Elimination

Local Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1</td>
<td>∅</td>
<td>{a, b, c, n}</td>
</tr>
<tr>
<td>n2</td>
<td>{a, n}</td>
<td>∅</td>
</tr>
<tr>
<td>n3</td>
<td>{a}</td>
<td>{a}</td>
</tr>
<tr>
<td>n4</td>
<td>{a}</td>
<td>∅</td>
</tr>
<tr>
<td>n5</td>
<td>{a, b, c}</td>
<td>{a, t1}</td>
</tr>
<tr>
<td>n6</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>

Global Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Iteration #1</th>
<th>Iteration #2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>n6</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>n5</td>
<td>∅</td>
<td>{a, b, c}</td>
</tr>
<tr>
<td>n4</td>
<td>{a, b, c}</td>
<td>{a, b, c}</td>
</tr>
<tr>
<td>n3</td>
<td>∅</td>
<td>{a}</td>
</tr>
<tr>
<td>n2</td>
<td>{a, b, c}</td>
<td>{a, b, c, n}</td>
</tr>
<tr>
<td>n1</td>
<td>{a, b, c, n}</td>
<td>∅</td>
</tr>
</tbody>
</table>
Tutorial Problem 1: Round #2 of Dead Code Elimination

```
a = 4
b = 2
c = 3
n = c*2
```

```
if (a > n)
  n2
```

```
  n3
  a = a+1
```

```
if (a ≥ 12)
  n4
```

```
  t1 = a+b
  print "Hi"
```

```
print "Hello"
```

### Local Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1</td>
<td>∅</td>
<td>{a, b, c, n}</td>
</tr>
<tr>
<td>n2</td>
<td>{a, n}</td>
<td>∅</td>
</tr>
<tr>
<td>n3</td>
<td>{a}</td>
<td>{a}</td>
</tr>
<tr>
<td>n4</td>
<td>{a}</td>
<td>∅</td>
</tr>
<tr>
<td>n5</td>
<td>{a, b}</td>
<td>{t1}</td>
</tr>
<tr>
<td>n6</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>
**Tutorial Problem 1: Round #2 of Dead Code Elimination**

```plaintext
a = 4
b = 2
c = 3
n = c*2

if (a > n) n2
   T
   n3
   a = a+1
F

if (a ≥ 12) n4
   T
   n5
t1 = a+b
   print "Hi"
F

print "Hello" n6
```

### Local Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1</td>
<td>∅</td>
<td>{a, b, c, n}</td>
</tr>
<tr>
<td>n2</td>
<td>{a, n}</td>
<td>∅</td>
</tr>
<tr>
<td>n3</td>
<td>{a}</td>
<td>{a}</td>
</tr>
<tr>
<td>n4</td>
<td>{a}</td>
<td>∅</td>
</tr>
<tr>
<td>n5</td>
<td>{a, b}</td>
<td>{t1}</td>
</tr>
<tr>
<td>n6</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>

### Global Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Iteration #1</th>
<th>Iteration #2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>n6</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>n5</td>
<td>∅</td>
<td>{a, b}</td>
</tr>
<tr>
<td>n4</td>
<td>{a, b}</td>
<td>{a, b}</td>
</tr>
<tr>
<td>n3</td>
<td>∅</td>
<td>{a}</td>
</tr>
<tr>
<td>n2</td>
<td>{a, b}</td>
<td>{a, b, n}</td>
</tr>
<tr>
<td>n1</td>
<td>{a, b, n}</td>
<td>∅</td>
</tr>
</tbody>
</table>
Tutorial Problem 1: Round #2 of Dead Code Elimination

Local Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1</td>
<td>∅</td>
<td>{a, b, c, n}</td>
</tr>
<tr>
<td>n2</td>
<td>{a, n}</td>
<td>∅</td>
</tr>
<tr>
<td>n3</td>
<td>{a}</td>
<td>{a}</td>
</tr>
<tr>
<td>n4</td>
<td>{a}</td>
<td>∅</td>
</tr>
<tr>
<td>n5</td>
<td>{a, b}</td>
<td>{t1}</td>
</tr>
<tr>
<td>n6</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>

Global Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Iteration #1</th>
<th>Iteration #2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>n6</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>n5</td>
<td>∅</td>
<td>{a, b}</td>
</tr>
<tr>
<td>n4</td>
<td>{a, b}</td>
<td>{a, b}</td>
</tr>
<tr>
<td>n3</td>
<td>∅</td>
<td>{a}</td>
</tr>
<tr>
<td>n2</td>
<td>{a, b}</td>
<td>{a, b, n}</td>
</tr>
<tr>
<td>n1</td>
<td>{a, b, n}</td>
<td>∅</td>
</tr>
</tbody>
</table>
**Tutorial Problem 1: Round #2 of Dead Code Elimination**

```
a = 4
b = 2
c = 3
n = c*2
if (a > n)
    n2
F
    a = a+1
T
if (a ≥ 12)
    n4
F
    t1 = a+b
T
print "Hi"
print "Hello"
```

### Local Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1</td>
<td>∅</td>
<td>{a, b, c, n}</td>
</tr>
<tr>
<td>n2</td>
<td>{a, n}</td>
<td>∅</td>
</tr>
<tr>
<td>n3</td>
<td>{a}</td>
<td>{a}</td>
</tr>
<tr>
<td>n4</td>
<td>{a}</td>
<td>∅</td>
</tr>
<tr>
<td>n5</td>
<td>{a, b}</td>
<td>{t1}</td>
</tr>
<tr>
<td>n6</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>

### Global Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Iteration #1</th>
<th>Iteration #2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>n6</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>n5</td>
<td>∅</td>
<td>{a, b}</td>
</tr>
<tr>
<td>n4</td>
<td>{a, b}</td>
<td>{a, b}</td>
</tr>
<tr>
<td>n3</td>
<td>∅</td>
<td>{a}</td>
</tr>
<tr>
<td>n2</td>
<td>{a, b}</td>
<td>{a, b, n}</td>
</tr>
<tr>
<td>n1</td>
<td>{a, b, n}</td>
<td>∅</td>
</tr>
</tbody>
</table>
Tutorial Problem 1: Round #3 of Dead Code Elimination

```plaintext
a = 4
b = 2
c = 3
n = c*2
if (a > n)
a = a + 1
if (a ≥ 12)
print "Hi"
print "Hello"
```

Local Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1</td>
<td>∅</td>
<td>{a, b, c, n}</td>
</tr>
<tr>
<td>n2</td>
<td>{a, n}</td>
<td>∅</td>
</tr>
<tr>
<td>n3</td>
<td>{a}</td>
<td>{a}</td>
</tr>
<tr>
<td>n4</td>
<td>{a}</td>
<td>∅</td>
</tr>
<tr>
<td>n5</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>n6</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>
Tutorial Problem 1: Round #3 of Dead Code Elimination

```
a = 4
b = 2
c = 3
n = c*2
if (a > n)
    a = a + 1
if (a ≥ 12)
    print "Hi"
print "Hello"
```

Local Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1</td>
<td>∅</td>
<td>{a, b, c, n}</td>
</tr>
<tr>
<td>n2</td>
<td>{a, n}</td>
<td>∅</td>
</tr>
<tr>
<td>n3</td>
<td>{a}</td>
<td>{a}</td>
</tr>
<tr>
<td>n4</td>
<td>{a}</td>
<td>∅</td>
</tr>
<tr>
<td>n5</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>n6</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>

Global Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Iteration #1</th>
<th>Iteration #2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>n6</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>n5</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>n4</td>
<td>∅</td>
<td>{a}</td>
</tr>
<tr>
<td>n3</td>
<td>∅</td>
<td>{a}</td>
</tr>
<tr>
<td>n2</td>
<td>{a}</td>
<td>{a, n}</td>
</tr>
<tr>
<td>n1</td>
<td>{a, n}</td>
<td>∅</td>
</tr>
</tbody>
</table>
Tutorial Problem 1: Round #3 of Dead Code Elimination

Local Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1</td>
<td>∅</td>
<td>{a, b, c, n}</td>
</tr>
<tr>
<td>n2</td>
<td>{a, n}</td>
<td>∅</td>
</tr>
<tr>
<td>n3</td>
<td>{a}</td>
<td>{a}</td>
</tr>
<tr>
<td>n4</td>
<td>{a}</td>
<td>∅</td>
</tr>
<tr>
<td>n5</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>n6</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>

Global Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Iteration #1</th>
<th>Iteration #2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>n6</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>n5</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>n4</td>
<td>∅</td>
<td>{a}</td>
</tr>
<tr>
<td>n3</td>
<td>∅</td>
<td>{a}</td>
</tr>
<tr>
<td>n2</td>
<td>{a}</td>
<td>{a, n}</td>
</tr>
<tr>
<td>n1</td>
<td>{a, n}</td>
<td>∅</td>
</tr>
</tbody>
</table>

a = 4
b = 2
c = 3
n = c*2

if (a > n)
    n2
    T
    F
    n3
    a = a + 1

if (a ≥ 12)
    n4
    T
    F
    n5
    print "Hi"

print "Hello"
Tutorial Problem 1: Round #3 of Dead Code Elimination

Local Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1</td>
<td>∅</td>
<td>{a, b, c, n}</td>
</tr>
<tr>
<td>n2</td>
<td>{a, n}</td>
<td>∅</td>
</tr>
<tr>
<td>n3</td>
<td>{a}</td>
<td>{a}</td>
</tr>
<tr>
<td>n4</td>
<td>{a}</td>
<td>∅</td>
</tr>
<tr>
<td>n5</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>n6</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>

Global Data Flow Information

<table>
<thead>
<tr>
<th></th>
<th>Iteration #1</th>
<th>Iteration #2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>n6</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>n5</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>n4</td>
<td>∅</td>
<td>{a}</td>
</tr>
<tr>
<td>n3</td>
<td>∅</td>
<td>{a}</td>
</tr>
<tr>
<td>n2</td>
<td>{a}</td>
<td>{a, n}</td>
</tr>
<tr>
<td>n1</td>
<td>{a, n}</td>
<td>∅</td>
</tr>
</tbody>
</table>

Local Data Flow Graph:

- a = 4
- b = 2
- c = 3
- n = c*2

```
if (a > n)
  a = a + 1
```

```
if (a ≥ 12)
  print "Hi"
```

```
print "Hello"
```

Jul 2017

IIT Bombay
Part 3

Some Observations
What Does Data Flow Analysis Involve?

- Defining the analysis.
- Formulating the analysis.
- Performing the analysis.
What Does Data Flow Analysis Involve?

- Defining the analysis. Define the properties of execution paths
- Formulating the analysis.
- Performing the analysis.
What Does Data Flow Analysis Involve?

- **Defining the analysis.** Define the properties of execution paths
- **Formulating the analysis.** Define data flow equations
  - Linear simultaneous equations on sets rather than numbers
  - Later we will generalize the domain of values
- **Performing the analysis.**
What Does Data Flow Analysis Involve?

- **Defining the analysis.** Define the properties of execution paths
- **Formulating the analysis.** Define data flow equations
  - Linear simultaneous equations on sets rather than numbers
  - Later we will generalize the domain of values
- **Performing the analysis.** Solve data flow equations for the given program flow graph
What Does Data Flow Analysis Involve?

- **Defining the analysis.** Define the properties of execution paths
- **Formulating the analysis.** Define data flow equations
  - Linear simultaneous equations on sets rather than numbers
  - Later we will generalize the domain of values
- **Performing the analysis.** Solve data flow equations for the given program flow graph
- Many unanswered questions
A Digression: Iterative Solution of Linear Simultaneous Equations

- Simultaneous equations represented in the form of the product of a matrix of coefficients ($A$) with the vector of unknowns ($x$)

$$Ax = b$$

- Start with approximate values
- Compute new values repeatedly from old values
- Two classical methods
  - Gauss-Seidel Method (Gauss: 1823, 1826), (Seidel: 1874)
  - Jacobi Method (Jacobi: 1845)
A Digression: An Example of Iterative Solution of Linear Simultaneous Equations

<table>
<thead>
<tr>
<th>Equations</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>4w = x + y + 32</td>
<td></td>
</tr>
<tr>
<td>4x = y + z + 32</td>
<td></td>
</tr>
<tr>
<td>4y = z + w + 32</td>
<td></td>
</tr>
<tr>
<td>4z = w + x + 32</td>
<td></td>
</tr>
<tr>
<td>w = x = y = z = 16</td>
<td></td>
</tr>
</tbody>
</table>

- Rewrite the equations to define $w$, $x$, $y$, and $z$
  
  $w = 0.25x + 0.25y + 8$
  $x = 0.25y + 0.25z + 8$
  $y = 0.25z + 0.25w + 8$
  $z = 0.25w + 0.25x + 8$

- Assume some initial values of $w_0$, $x_0$, $y_0$, and $z_0$

- Compute $w_i$, $x_i$, $y_i$, and $z_i$ within some margin of error
### A Digression: Gauss-Seidel Method

#### Equations

<table>
<thead>
<tr>
<th>Equation</th>
<th>Initial Values</th>
<th>Error Margin</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w = 0.25x + 0.25y + 8 )</td>
<td>( w_0 = 24 )</td>
<td>( w_{i+1} - w_i \leq 0.35 )</td>
</tr>
<tr>
<td>( x = 0.25y + 0.25z + 8 )</td>
<td>( x_0 = 24 )</td>
<td>( x_{i+1} - x_i \leq 0.35 )</td>
</tr>
<tr>
<td>( y = 0.25z + 0.25w + 8 )</td>
<td>( y_0 = 24 )</td>
<td>( y_{i+1} - y_i \leq 0.35 )</td>
</tr>
<tr>
<td>( z = 0.25w + 0.25x + 8 )</td>
<td>( z_0 = 24 )</td>
<td>( z_{i+1} - z_i \leq 0.35 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th>Iteration 2</th>
<th>Iteration 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration 4</th>
<th>Iteration 5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### A Digression: Gauss-Seidel Method

#### Equations

<table>
<thead>
<tr>
<th>Equation</th>
<th>Initial Values</th>
<th>Error Margin</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w = 0.25x + 0.25y + 8 )</td>
<td>( w_0 = 24 )</td>
<td>( w_{i+1} - w_i \leq 0.35 )</td>
</tr>
<tr>
<td>( x = 0.25y + 0.25z + 8 )</td>
<td>( x_0 = 24 )</td>
<td>( x_{i+1} - x_i \leq 0.35 )</td>
</tr>
<tr>
<td>( y = 0.25z + 0.25w + 8 )</td>
<td>( y_0 = 24 )</td>
<td>( y_{i+1} - y_i \leq 0.35 )</td>
</tr>
<tr>
<td>( z = 0.25w + 0.25x + 8 )</td>
<td>( z_0 = 24 )</td>
<td>( z_{i+1} - z_i \leq 0.35 )</td>
</tr>
</tbody>
</table>

#### Iterations

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th>Iteration 2</th>
<th>Iteration 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w_1 = 6 + 6 + 8 = 20 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( x_1 = 6 + 6 + 8 = 20 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( y_1 = 6 + 6 + 8 = 20 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( z_1 = 6 + 6 + 8 = 20 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Iterations 4 and 5

<table>
<thead>
<tr>
<th>Iteration 4</th>
<th>Iteration 5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# A Digression: Gauss-Seidel Method

<table>
<thead>
<tr>
<th>Equations</th>
<th>Initial Values</th>
<th>Error Margin</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w = 0.25x + 0.25y + 8 )</td>
<td>( w_0 = 24 )</td>
<td>( w_{i+1} - w_i \leq 0.35 )</td>
</tr>
<tr>
<td>( x = 0.25y + 0.25z + 8 )</td>
<td>( x_0 = 24 )</td>
<td>( x_{i+1} - x_i \leq 0.35 )</td>
</tr>
<tr>
<td>( y = 0.25z + 0.25w + 8 )</td>
<td>( y_0 = 24 )</td>
<td>( y_{i+1} - y_i \leq 0.35 )</td>
</tr>
<tr>
<td>( z = 0.25w + 0.25x + 8 )</td>
<td>( z_0 = 24 )</td>
<td>( z_{i+1} - z_i \leq 0.35 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th>Iteration 2</th>
<th>Iteration 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w_1 = 6 + 6 + 8 = 20 )</td>
<td>( w_2 = 5 + 5 + 8 = 18 )</td>
<td></td>
</tr>
<tr>
<td>( x_1 = 6 + 6 + 8 = 20 )</td>
<td>( x_2 = 5 + 5 + 8 = 18 )</td>
<td></td>
</tr>
<tr>
<td>( y_1 = 6 + 6 + 8 = 20 )</td>
<td>( y_2 = 5 + 5 + 8 = 18 )</td>
<td></td>
</tr>
<tr>
<td>( z_1 = 6 + 6 + 8 = 20 )</td>
<td>( z_2 = 5 + 5 + 8 = 18 )</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration 4</th>
<th>Iteration 5</th>
</tr>
</thead>
</table>
A Digression: Gauss-Seidel Method

Equations

<table>
<thead>
<tr>
<th>w = 0.25x + 0.25y + 8</th>
<th>x = 0.25y + 0.25z + 8</th>
<th>y = 0.25z + 0.25w + 8</th>
<th>z = 0.25w + 0.25x + 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>w0 = 24</td>
<td>x0 = 24</td>
<td>y0 = 24</td>
<td>z0 = 24</td>
</tr>
<tr>
<td>Error Margin</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wi+1 − wi ≤ 0.35</td>
<td>xi+1 − xi ≤ 0.35</td>
<td>yi+1 − yi ≤ 0.35</td>
<td>zi+1 − zi ≤ 0.35</td>
</tr>
</tbody>
</table>

Iteration 1

<table>
<thead>
<tr>
<th>w1 = 6 + 6 + 8 = 20</th>
<th>x1 = 6 + 6 + 8 = 20</th>
<th>y1 = 6 + 6 + 8 = 20</th>
<th>z1 = 6 + 6 + 8 = 20</th>
</tr>
</thead>
</table>

Iteration 2

<table>
<thead>
<tr>
<th>w2 = 5 + 5 + 8 = 18</th>
<th>x2 = 5 + 5 + 8 = 18</th>
<th>y2 = 5 + 5 + 8 = 18</th>
<th>z2 = 5 + 5 + 8 = 18</th>
</tr>
</thead>
</table>

Iteration 3

<table>
<thead>
<tr>
<th>w3 = 4.5 + 4.5 + 8 = 17</th>
<th>x3 = 4.5 + 4.5 + 8 = 17</th>
<th>y3 = 4.5 + 4.5 + 8 = 17</th>
<th>z3 = 4.5 + 4.5 + 8 = 17</th>
</tr>
</thead>
</table>

Iteration 4

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>

Iteration 5

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
## A Digression: Gauss-Seidel Method

<table>
<thead>
<tr>
<th>Equations</th>
<th>Initial Values</th>
<th>Error Margin</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w = 0.25x + 0.25y + 8$</td>
<td>$w_0 = 24$</td>
<td>$w_{i+1} - w_i \leq 0.35$</td>
</tr>
<tr>
<td>$x = 0.25y + 0.25z + 8$</td>
<td>$x_0 = 24$</td>
<td>$x_{i+1} - x_i \leq 0.35$</td>
</tr>
<tr>
<td>$y = 0.25z + 0.25w + 8$</td>
<td>$y_0 = 24$</td>
<td>$y_{i+1} - y_i \leq 0.35$</td>
</tr>
<tr>
<td>$z = 0.25w + 0.25x + 8$</td>
<td>$z_0 = 24$</td>
<td>$z_{i+1} - z_i \leq 0.35$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th>Iteration 2</th>
<th>Iteration 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1 = 6 + 6 + 8 = 20$</td>
<td>$w_2 = 5 + 5 + 8 = 18$</td>
<td>$w_3 = 4.5 + 4.5 + 8 = 17$</td>
</tr>
<tr>
<td>$x_1 = 6 + 6 + 8 = 20$</td>
<td>$x_2 = 5 + 5 + 8 = 18$</td>
<td>$x_3 = 4.5 + 4.5 + 8 = 17$</td>
</tr>
<tr>
<td>$y_1 = 6 + 6 + 8 = 20$</td>
<td>$y_2 = 5 + 5 + 8 = 18$</td>
<td>$y_3 = 4.5 + 4.5 + 8 = 17$</td>
</tr>
<tr>
<td>$z_1 = 6 + 6 + 8 = 20$</td>
<td>$z_2 = 5 + 5 + 8 = 18$</td>
<td>$z_3 = 4.5 + 4.5 + 8 = 17$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration 4</th>
<th>Iteration 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_4 = 4.25 + 4.25 + 8 = 16.5$</td>
<td></td>
</tr>
<tr>
<td>$x_4 = 4.25 + 4.25 + 8 = 16.5$</td>
<td></td>
</tr>
<tr>
<td>$y_4 = 4.25 + 4.25 + 8 = 16.5$</td>
<td></td>
</tr>
<tr>
<td>$z_4 = 4.25 + 4.25 + 8 = 16.5$</td>
<td></td>
</tr>
</tbody>
</table>
## A Digression: Gauss-Seidel Method

### Equations

<table>
<thead>
<tr>
<th>Variable</th>
<th>Equation</th>
<th>Initial Value</th>
<th>Error Margin</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w$</td>
<td>$0.25x + 0.25y + 8$</td>
<td>$w_0 = 24$</td>
<td>$w_{i+1} - w_i \leq 0.35$</td>
</tr>
<tr>
<td>$x$</td>
<td>$0.25y + 0.25z + 8$</td>
<td>$x_0 = 24$</td>
<td>$x_{i+1} - x_i \leq 0.35$</td>
</tr>
<tr>
<td>$y$</td>
<td>$0.25z + 0.25w + 8$</td>
<td>$y_0 = 24$</td>
<td>$y_{i+1} - y_i \leq 0.35$</td>
</tr>
<tr>
<td>$z$</td>
<td>$0.25w + 0.25x + 8$</td>
<td>$z_0 = 24$</td>
<td>$z_{i+1} - z_i \leq 0.35$</td>
</tr>
</tbody>
</table>

### Initial Values

<table>
<thead>
<tr>
<th>$w_0$</th>
<th>$x_0$</th>
<th>$y_0$</th>
<th>$z_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
</tr>
</tbody>
</table>

### Error Margin

$w_{i+1} - w_i \leq 0.35$

$w_{i+1} - w_i \leq 0.35$

$w_{i+1} - w_i \leq 0.35$

### Iteration Table

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$6 + 6 + 8 = 20$</td>
<td>$5 + 5 + 8 = 18$</td>
<td>$4.5 + 4.5 + 8 = 17$</td>
</tr>
<tr>
<td>2</td>
<td>$6 + 6 + 8 = 20$</td>
<td>$5 + 5 + 8 = 18$</td>
<td>$4.5 + 4.5 + 8 = 17$</td>
</tr>
<tr>
<td>3</td>
<td>$6 + 6 + 8 = 20$</td>
<td>$5 + 5 + 8 = 18$</td>
<td>$4.5 + 4.5 + 8 = 17$</td>
</tr>
<tr>
<td>4</td>
<td>$6 + 6 + 8 = 20$</td>
<td>$5 + 5 + 8 = 18$</td>
<td>$4.5 + 4.5 + 8 = 17$</td>
</tr>
<tr>
<td>5</td>
<td>$6 + 6 + 8 = 20$</td>
<td>$5 + 5 + 8 = 18$</td>
<td>$4.5 + 4.5 + 8 = 17$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$6 + 6 + 8 = 20$</td>
<td>$5 + 5 + 8 = 18$</td>
<td>$4.5 + 4.5 + 8 = 17$</td>
</tr>
<tr>
<td>2</td>
<td>$6 + 6 + 8 = 20$</td>
<td>$5 + 5 + 8 = 18$</td>
<td>$4.5 + 4.5 + 8 = 17$</td>
</tr>
<tr>
<td>3</td>
<td>$6 + 6 + 8 = 20$</td>
<td>$5 + 5 + 8 = 18$</td>
<td>$4.5 + 4.5 + 8 = 17$</td>
</tr>
<tr>
<td>4</td>
<td>$6 + 6 + 8 = 20$</td>
<td>$5 + 5 + 8 = 18$</td>
<td>$4.5 + 4.5 + 8 = 17$</td>
</tr>
<tr>
<td>5</td>
<td>$6 + 6 + 8 = 20$</td>
<td>$5 + 5 + 8 = 18$</td>
<td>$4.5 + 4.5 + 8 = 17$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$y_1$</th>
<th>$y_2$</th>
<th>$y_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$6 + 6 + 8 = 20$</td>
<td>$5 + 5 + 8 = 18$</td>
<td>$4.5 + 4.5 + 8 = 17$</td>
</tr>
<tr>
<td>2</td>
<td>$6 + 6 + 8 = 20$</td>
<td>$5 + 5 + 8 = 18$</td>
<td>$4.5 + 4.5 + 8 = 17$</td>
</tr>
<tr>
<td>3</td>
<td>$6 + 6 + 8 = 20$</td>
<td>$5 + 5 + 8 = 18$</td>
<td>$4.5 + 4.5 + 8 = 17$</td>
</tr>
<tr>
<td>4</td>
<td>$6 + 6 + 8 = 20$</td>
<td>$5 + 5 + 8 = 18$</td>
<td>$4.5 + 4.5 + 8 = 17$</td>
</tr>
<tr>
<td>5</td>
<td>$6 + 6 + 8 = 20$</td>
<td>$5 + 5 + 8 = 18$</td>
<td>$4.5 + 4.5 + 8 = 17$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$z_1$</th>
<th>$z_2$</th>
<th>$z_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$6 + 6 + 8 = 20$</td>
<td>$5 + 5 + 8 = 18$</td>
<td>$4.5 + 4.5 + 8 = 17$</td>
</tr>
<tr>
<td>2</td>
<td>$6 + 6 + 8 = 20$</td>
<td>$5 + 5 + 8 = 18$</td>
<td>$4.5 + 4.5 + 8 = 17$</td>
</tr>
<tr>
<td>3</td>
<td>$6 + 6 + 8 = 20$</td>
<td>$5 + 5 + 8 = 18$</td>
<td>$4.5 + 4.5 + 8 = 17$</td>
</tr>
<tr>
<td>4</td>
<td>$6 + 6 + 8 = 20$</td>
<td>$5 + 5 + 8 = 18$</td>
<td>$4.5 + 4.5 + 8 = 17$</td>
</tr>
<tr>
<td>5</td>
<td>$6 + 6 + 8 = 20$</td>
<td>$5 + 5 + 8 = 18$</td>
<td>$4.5 + 4.5 + 8 = 17$</td>
</tr>
</tbody>
</table>

### Continued

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$w_4$</th>
<th>$w_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>$4.25 + 4.25 + 8 = 16.5$</td>
<td>$4.125 + 4.125 + 8 = 16.25$</td>
</tr>
<tr>
<td>5</td>
<td>$4.25 + 4.25 + 8 = 16.5$</td>
<td>$4.125 + 4.125 + 8 = 16.25$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$x_4$</th>
<th>$x_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>$4.25 + 4.25 + 8 = 16.5$</td>
<td>$4.125 + 4.125 + 8 = 16.25$</td>
</tr>
<tr>
<td>5</td>
<td>$4.25 + 4.25 + 8 = 16.5$</td>
<td>$4.125 + 4.125 + 8 = 16.25$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$y_4$</th>
<th>$y_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>$4.25 + 4.25 + 8 = 16.5$</td>
<td>$4.125 + 4.125 + 8 = 16.25$</td>
</tr>
<tr>
<td>5</td>
<td>$4.25 + 4.25 + 8 = 16.5$</td>
<td>$4.125 + 4.125 + 8 = 16.25$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$z_4$</th>
<th>$z_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>$4.25 + 4.25 + 8 = 16.5$</td>
<td>$4.125 + 4.125 + 8 = 16.25$</td>
</tr>
<tr>
<td>5</td>
<td>$4.25 + 4.25 + 8 = 16.5$</td>
<td>$4.125 + 4.125 + 8 = 16.25$</td>
</tr>
</tbody>
</table>
A Digression: Jacobi Method

Use values from the current iteration wherever possible

<table>
<thead>
<tr>
<th>Equations</th>
<th>Initial Values</th>
<th>Error Margin</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w = 0.25x + 0.25y + 8$</td>
<td>$w_0 = 24$</td>
<td>$w_{i+1} - w_i \leq 0.35$</td>
</tr>
<tr>
<td>$x = 0.25y + 0.25z + 8$</td>
<td>$x_0 = 24$</td>
<td>$x_{i+1} - x_i \leq 0.35$</td>
</tr>
<tr>
<td>$y = 0.25z + 0.25w + 8$</td>
<td>$y_0 = 24$</td>
<td>$y_{i+1} - y_i \leq 0.35$</td>
</tr>
<tr>
<td>$z = 0.25w + 0.25x + 8$</td>
<td>$z_0 = 24$</td>
<td>$z_{i+1} - z_i \leq 0.35$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th>Iteration 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration 3</th>
<th>Iteration 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A Digression: Jacobi Method

Use values from the current iteration wherever possible

<table>
<thead>
<tr>
<th>Equations</th>
<th>Initial Values</th>
<th>Error Margin</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w = 0.25x + 0.25y + 8 )</td>
<td>( w_0 = 24 )</td>
<td>( w_{i+1} - w_i \leq 0.35 )</td>
</tr>
<tr>
<td>( x = 0.25y + 0.25z + 8 )</td>
<td>( x_0 = 24 )</td>
<td>( x_{i+1} - x_i \leq 0.35 )</td>
</tr>
<tr>
<td>( y = 0.25z + 0.25w + 8 )</td>
<td>( y_0 = 24 )</td>
<td>( y_{i+1} - y_i \leq 0.35 )</td>
</tr>
<tr>
<td>( z = 0.25w + 0.25x + 8 )</td>
<td>( z_0 = 24 )</td>
<td>( z_{i+1} - z_i \leq 0.35 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th>Iteration 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w_1 = 6 + 6 + 8 = 20 )</td>
<td></td>
</tr>
<tr>
<td>( x_1 = 6 + 6 + 8 = 20 )</td>
<td></td>
</tr>
<tr>
<td>( y_1 = 6 + 5 + 8 = 19 )</td>
<td></td>
</tr>
<tr>
<td>( z_1 = 5 + 5 + 8 = 18 )</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration 3</th>
<th>Iteration 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A Digression: Jacobi Method

Use values from the current iteration wherever possible

<table>
<thead>
<tr>
<th>Equations</th>
<th>Initial Values</th>
<th>Error Margin</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w = 0.25x + 0.25y + 8 )</td>
<td>( w_0 = 24 )</td>
<td>( w_{i+1} - w_i \leq 0.35 )</td>
</tr>
<tr>
<td>( x = 0.25y + 0.25z + 8 )</td>
<td>( x_0 = 24 )</td>
<td>( x_{i+1} - x_i \leq 0.35 )</td>
</tr>
<tr>
<td>( y = 0.25z + 0.25w + 8 )</td>
<td>( y_0 = 24 )</td>
<td>( y_{i+1} - y_i \leq 0.35 )</td>
</tr>
<tr>
<td>( z = 0.25w + 0.25x + 8 )</td>
<td>( z_0 = 24 )</td>
<td>( z_{i+1} - z_i \leq 0.35 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th>Iteration 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w_1 = 6 + 6 + 8 = 20 )</td>
<td>( w_2 = 5 + 4.75 + 8 = 17.75 )</td>
</tr>
<tr>
<td>( x_1 = 6 + 6 + 8 = 20 )</td>
<td>( x_2 = 4.75 + 4.5 + 8 = 17.25 )</td>
</tr>
<tr>
<td>( y_1 = 6 + 5 + 8 = 19 )</td>
<td>( y_2 = 4.5 + 4.4375 + 8 = 16.935 )</td>
</tr>
<tr>
<td>( z_1 = 5 + 5 + 8 = 18 )</td>
<td>( z_2 = 4.4375 + 4.375 + 8 = 16.8125 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration 3</th>
<th>Iteration 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A Digression: Jacobi Method

Use values from the current iteration wherever possible

<table>
<thead>
<tr>
<th>Equations</th>
<th>Initial Values</th>
<th>Error Margin</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w = 0.25x + 0.25y + 8$</td>
<td>$w_0 = 24$</td>
<td>$w_{i+1} - w_i \leq 0.35$</td>
</tr>
<tr>
<td>$x = 0.25y + 0.25z + 8$</td>
<td>$x_0 = 24$</td>
<td>$x_{i+1} - x_i \leq 0.35$</td>
</tr>
<tr>
<td>$y = 0.25z + 0.25w + 8$</td>
<td>$y_0 = 24$</td>
<td>$y_{i+1} - y_i \leq 0.35$</td>
</tr>
<tr>
<td>$z = 0.25w + 0.25x + 8$</td>
<td>$z_0 = 24$</td>
<td>$z_{i+1} - z_i \leq 0.35$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th>Iteration 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1 = 6 + 6 + 8 = 20$</td>
<td>$w_2 = 5 + 4.75 + 8 = 17.75$</td>
</tr>
<tr>
<td>$x_1 = 6 + 6 + 8 = 20$</td>
<td>$x_2 = 4.75 + 4.5 + 8 = 17.25$</td>
</tr>
<tr>
<td>$y_1 = 6 + 5 + 8 = 19$</td>
<td>$y_2 = 4.5 + 4.4375 + 8 = 16.935$</td>
</tr>
<tr>
<td>$z_1 = 5 + 5 + 8 = 18$</td>
<td>$z_2 = 4.4375 + 4.375 + 8 = 16.8125$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration 3</th>
<th>Iteration 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_3 = 4.3125 + 4.23375 + 8 = 16.54625$</td>
<td></td>
</tr>
<tr>
<td>$x_3 = 4.23375 + 4.23375 + 8 = 16.436875$</td>
<td></td>
</tr>
<tr>
<td>$y_3 = 4.23375 + 4.1365625 + 8 = 16.370$</td>
<td></td>
</tr>
<tr>
<td>$z_3 = 4.1365625 + 4.11 + 8 = 16.34375$</td>
<td></td>
</tr>
</tbody>
</table>
### A Digression: Jacobi Method

Use values from the current iteration wherever possible

<table>
<thead>
<tr>
<th>Equations</th>
<th>Initial Values</th>
<th>Error Margin</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w = 0.25x + 0.25y + 8 )</td>
<td>( w_0 = 24 )</td>
<td>( w_{i+1} - w_i \leq 0.35 )</td>
</tr>
<tr>
<td>( x = 0.25y + 0.25z + 8 )</td>
<td>( x_0 = 24 )</td>
<td>( x_{i+1} - x_i \leq 0.35 )</td>
</tr>
<tr>
<td>( y = 0.25z + 0.25w + 8 )</td>
<td>( y_0 = 24 )</td>
<td>( y_{i+1} - y_i \leq 0.35 )</td>
</tr>
<tr>
<td>( z = 0.25w + 0.25x + 8 )</td>
<td>( z_0 = 24 )</td>
<td>( z_{i+1} - z_i \leq 0.35 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th>Iteration 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w_1 = 6 + 6 + 8 = 20 )</td>
<td>( w_2 = 5 + 4.75 + 8 = 17.75 )</td>
</tr>
<tr>
<td>( x_1 = 6 + 6 + 8 = 20 )</td>
<td>( x_2 = 4.75 + 4.5 + 8 = 17.25 )</td>
</tr>
<tr>
<td>( y_1 = 6 + 5 + 8 = 19 )</td>
<td>( y_2 = 4.5 + 4.4375 + 8 = 16.935 )</td>
</tr>
<tr>
<td>( z_1 = 5 + 5 + 8 = 18 )</td>
<td>( z_2 = 4.4375 + 4.375 + 8 = 16.8125 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration 3</th>
<th>Iteration 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w_3 = 4.3125 + 4.23375 + 8 = 16.54625 )</td>
<td>( w_4 = 16.20172 )</td>
</tr>
<tr>
<td>( x_3 = 4.23375 + 4.23375 + 8 = 16.436875 )</td>
<td>( x_4 = 16.17844 )</td>
</tr>
<tr>
<td>( y_3 = 4.23375 + 4.1365625 + 8 = 16.370 )</td>
<td>( y_4 = 16.13637 )</td>
</tr>
<tr>
<td>( z_3 = 4.1365625 + 4.11 + 8 = 16.34375 )</td>
<td>( z_4 = 16.09504 )</td>
</tr>
</tbody>
</table>
Our Method of Performing Data Flow Analysis

• Round robin iteration
• Essentially Jacobi method
• Unknowns are the data flow variables $In_i$ and $Out_i$
• Domain of values is not numbers
• Computation in a fixed order
  ▶ either forward (reverse post order) traversal, or
  ▶ backward (post order) traversal

over the control flow graph
Tutorial Problem 2 for Liveness Analysis

Draw the control flow graph and perform live variables analysis

```c
int f(int m, int n, int k)
{
    int a,i;

    for (i=m-1; i<k; i++)
    {
        if (i>=n)
        {
            a = n;
            a = a+i;
        }
    return a;
    }
```
Tutorial Problem 2 for Liveness Analysis

Draw the control flow graph and perform live variables analysis

```c
int f(int m, int n, int k)
{
    int a, i;

    for (i=m-1; i<k; i++)
    {
        if (i>=n)
            a = n;
        a = a+i;
    }

    return a;
}
```
The Semantics of Return Statement for Live Variables Analysis

“return a” is modelled by the statement “return_value_in_stack = a”

- If we assume that the statement is executed *within* the block

- If we assume that the statement is executed *outside of* the block and along the edge connecting the procedure to its caller
The Semantics of Return Statement for Live Variables Analysis

“return a” is modelled by the statement “return_value_in_stack = a”

- If we assume that the statement is executed within the block
  \[\Rightarrow BI \text{ can be } \emptyset\]

- If we assume that the statement is executed outside of the block and along the edge connecting the procedure to its caller
  \[\Rightarrow a \in BI\]
### Solution of Tutorial Problem 2

<table>
<thead>
<tr>
<th>Block</th>
<th>Local Information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gen(_n)</td>
<td>Kill(_n)</td>
</tr>
<tr>
<td>(n_6)</td>
<td>{a}</td>
<td>∅</td>
</tr>
<tr>
<td>(n_5)</td>
<td>{a, i}</td>
<td>{a, i}</td>
</tr>
<tr>
<td>(n_4)</td>
<td>{n}</td>
<td>{a}</td>
</tr>
<tr>
<td>(n_3)</td>
<td>{i, n}</td>
<td>∅</td>
</tr>
<tr>
<td>(n_2)</td>
<td>{i, k}</td>
<td>∅</td>
</tr>
<tr>
<td>(n_1)</td>
<td>{m}</td>
<td>{i}</td>
</tr>
</tbody>
</table>
## Solution of Tutorial Problem 2

<table>
<thead>
<tr>
<th>Block</th>
<th>Local Information</th>
<th>Global Information</th>
<th>Iteration # 1</th>
<th>Change in iteration # 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>$Gen_n$</td>
<td>$Kill_n$</td>
</tr>
<tr>
<td>$n_6$</td>
<td>${a}$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>${a}$</td>
</tr>
<tr>
<td>$n_5$</td>
<td>${a, i}$</td>
<td>${a, i}$</td>
<td>$\emptyset$</td>
<td>${a, i}$</td>
</tr>
<tr>
<td>$n_4$</td>
<td>${n}$</td>
<td>${a}$</td>
<td>${a, i}$</td>
<td>${i, n}$</td>
</tr>
<tr>
<td>$n_3$</td>
<td>${i, n}$</td>
<td>$\emptyset$</td>
<td>${a, i, n}$</td>
<td>${a, i, n}$</td>
</tr>
<tr>
<td>$n_2$</td>
<td>${i, k}$</td>
<td>$\emptyset$</td>
<td>${a, i, n}$</td>
<td>${a, i, k, n}$</td>
</tr>
<tr>
<td>$n_1$</td>
<td>${m}$</td>
<td>${i}$</td>
<td>${a, i, k, n}$</td>
<td>${a, k, m, n}$</td>
</tr>
</tbody>
</table>
# Solution of Tutorial Problem 2

<table>
<thead>
<tr>
<th>Block</th>
<th>Local Information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gen$_n$</td>
<td>Kill$_n$</td>
</tr>
<tr>
<td>$n_6$</td>
<td>${a}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$n_5$</td>
<td>${a, i}$</td>
<td>${a, i}$</td>
</tr>
<tr>
<td>$n_4$</td>
<td>${n}$</td>
<td>${a}$</td>
</tr>
<tr>
<td>$n_3$</td>
<td>${i, n}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$n_2$</td>
<td>${i, k}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$n_1$</td>
<td>${m}$</td>
<td>${i}$</td>
</tr>
</tbody>
</table>
Interpreting the Result of Liveness Analysis for Tutorial Problem 2

- Is a live at the exit of $n_5$ at the end of iteration 1? Why?
  (We have used post order traversal)
Interpreting the Result of Liveness Analysis for Tutorial Problem 2

- Is a live at the exit of $n_5$ at the end of iteration 1? Why?
  (We have used post order traversal)

- Is a live at the exit of $n_5$ at the end of iteration 2? Why?
  (We have used post order traversal)
Interpreting the Result of Liveness Analysis for Tutorial Problem 2

- Is a live at the exit of $n_5$ at the end of iteration 1? Why?
  (We have used post order traversal)

- Is a live at the exit of $n_5$ at the end of iteration 2? Why?
  (We have used post order traversal)

- Show an execution path along which $a$ is live at the exit of $n_5$
Interpreting the Result of Liveness Analysis for Tutorial Problem 2

- Is a live at the exit of $n_5$ at the end of iteration 1? Why?
  (We have used post order traversal)

- Is a live at the exit of $n_5$ at the end of iteration 2? Why?
  (We have used post order traversal)

- Show an execution path along which $a$ is live at the exit of $n_5$

- Show an execution path along which $a$ is live at the exit of $n_3$
Interpreting the Result of Liveness Analysis for Tutorial Problem 2

- Is a live at the exit of $n_5$ at the end of iteration 1? Why?
  (We have used post order traversal)
- Is a live at the exit of $n_5$ at the end of iteration 2? Why?
  (We have used post order traversal)
- Show an execution path along which $a$ is live at the exit of $n_5$
- Show an execution path along which $a$ is live at the exit of $n_3$

$n_1 \rightarrow n_2 \rightarrow n_3 \rightarrow n_5 \rightarrow n_2 \rightarrow \ldots$
Interpreting the Result of Liveness Analysis for Tutorial Problem 2

- Is a live at the exit of $n_5$ at the end of iteration 1? Why?
  (We have used post order traversal)
- Is a live at the exit of $n_5$ at the end of iteration 2? Why?
  (We have used post order traversal)
- Show an execution path along which $a$ is live at the exit of $n_5$
- Show an execution path along which $a$ is live at the exit of $n_3$
  $n_1 \rightarrow n_2 \rightarrow n_3 \rightarrow n_5 \rightarrow n_2 \rightarrow \ldots$
- Show an execution path along which $a$ is not live at the exit of $n_3$
Interpreting the Result of Liveness Analysis for Tutorial Problem 2

- Is a live at the exit of $n_5$ at the end of iteration 1? Why?
  (We have used post order traversal)

- Is a live at the exit of $n_5$ at the end of iteration 2? Why?
  (We have used post order traversal)

- Show an execution path along which $a$ is live at the exit of $n_5$

- Show an execution path along which $a$ is live at the exit of $n_3$

- Show an execution path along which $a$ is not live at the exit of $n_3$

\[ n_1 \rightarrow n_2 \rightarrow n_3 \rightarrow n_5 \rightarrow n_2 \rightarrow \ldots \]

\[ n_1 \rightarrow n_2 \rightarrow n_3 \rightarrow n_4 \rightarrow n_2 \rightarrow \ldots \]
Tutorial Problem 3 for Liveness Analysis

Also write a C program for this CFG without using goto or break

```
x = 1
y = 2

if (c)

x = y + 1
y = 2 * z
if (d)

x = y + z

z = 1
if (c < 20)

z = x
```
Tutorial Problem 3 for Liveness Analysis

Also write a C program for this CFG without using goto or break

```c
void f()
{
    int x, y, z;
    int c, d;
    x = 1;
    y = 2;
    if (c)
    {
        do
        {
            x = y+1;
            y = 2*z;
            if (d)
            {
                x = y+z;
                z = 1;
            }
        } while (c < 20);
    }
    z = x;
}
```
## Solution of Tutorial Problem 3

<table>
<thead>
<tr>
<th>Block</th>
<th>Local Information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Iteration # 1</td>
</tr>
<tr>
<td></td>
<td>Gen(_n)</td>
<td>Kill(_n)</td>
</tr>
<tr>
<td>(n_6)</td>
<td>{x}</td>
<td>{z}</td>
</tr>
<tr>
<td>(n_5)</td>
<td>{c}</td>
<td>{z}</td>
</tr>
<tr>
<td>(n_4)</td>
<td>{y, z}</td>
<td>{x}</td>
</tr>
<tr>
<td>(n_3)</td>
<td>{y, z, d}</td>
<td>{x, y}</td>
</tr>
<tr>
<td>(n_2)</td>
<td>{c}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>(n_1)</td>
<td>\emptyset</td>
<td>{x, y}</td>
</tr>
</tbody>
</table>
## Solution of Tutorial Problem 3

<table>
<thead>
<tr>
<th>Block</th>
<th>Local Information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( Gen_n )</td>
<td>( Kill_n )</td>
</tr>
<tr>
<td>( n_6 )</td>
<td>( { x } )</td>
<td>( { z } )</td>
</tr>
<tr>
<td>( n_5 )</td>
<td>( { c } )</td>
<td>( { z } )</td>
</tr>
<tr>
<td>( n_4 )</td>
<td>( { y, z } )</td>
<td>( { x } )</td>
</tr>
<tr>
<td>( n_3 )</td>
<td>( { y, z, d } )</td>
<td>( { x, y } )</td>
</tr>
<tr>
<td>( n_2 )</td>
<td>( { c } )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( n_1 )</td>
<td>( \emptyset )</td>
<td>( { x, y } )</td>
</tr>
</tbody>
</table>
## Solution of Tutorial Problem 3

<table>
<thead>
<tr>
<th>Block</th>
<th>Local Information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gen(_n)</td>
<td>Kill(_n)</td>
</tr>
<tr>
<td></td>
<td>Out(_n)</td>
<td>In(_n)</td>
</tr>
<tr>
<td>(n_6)</td>
<td>{x}</td>
<td>{z}</td>
</tr>
<tr>
<td>(n_5)</td>
<td>{c}</td>
<td>{z}</td>
</tr>
<tr>
<td>(n_4)</td>
<td>{y, z}</td>
<td>{x}</td>
</tr>
<tr>
<td>(n_3)</td>
<td>{y, z, d}</td>
<td>{x, y}</td>
</tr>
<tr>
<td>(n_2)</td>
<td>{c}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>(n_1)</td>
<td>\emptyset</td>
<td>{x, y}</td>
</tr>
</tbody>
</table>
Interpreting the Result of Liveness Analysis for Tutorial Problem 3

- Why is $z$ live at the exit of $n_5$?
Interpreting the Result of Liveness Analysis for Tutorial Problem 3

- Why is z live at the exit of $n_5$?
- Why is z not live at the entry of $n_5$?
Interpreting the Result of Liveness Analysis for Tutorial Problem 3

- Why is z live at the exit of n₅?
- Why is z not live at the entry of n₅?
- Why is x live at the exit of n₃ inspite of being killed in n₄?
Interpreting the Result of Liveness Analysis for Tutorial Problem 3

- Why is $z$ live at the exit of $n_5$?
- Why is $z$ not live at the entry of $n_5$?
- Why is $x$ live at the exit of $n_3$ inspite of being killed in $n_4$?
- Identify the instance of dead code elimination
Interpreting the Result of Liveness Analysis for Tutorial Problem 3

- Why is \( z \) live at the exit of \( n_5 \)?
- Why is \( z \) not live at the entry of \( n_5 \)?
- Why is \( x \) live at the exit of \( n_3 \) inspite of being killed in \( n_4 \)?
- Identify the instance of dead code elimination \( z \equiv x \) in \( n_6 \)
Interpreting the Result of Liveness Analysis for Tutorial Problem 3

- Why is $z$ live at the exit of $n_5$?
- Why is $z$ not live at the entry of $n_5$?
- Why is $x$ live at the exit of $n_3$ inspite of being killed in $n_4$?
- Identify the instance of dead code elimination $z = x$ in $n_6$
- Would the first round of dead code elimination cause liveness information to change?
Interpreting the Result of Liveness Analysis for Tutorial Problem 3

- Why is $z$ live at the exit of $n_5$?
- Why is $z$ not live at the entry of $n_5$?
- Why is $x$ live at the exit of $n_3$ inspite of being killed in $n_4$?
- Identify the instance of dead code elimination $z = x$ in $n_6$
- Would the first round of dead code elimination cause liveness information to change? Yes
Interpreting the Result of Liveness Analysis for Tutorial Problem 3

- Why is z live at the exit of $n_5$?
- Why is z not live at the entry of $n_5$?
- Why is x live at the exit of $n_3$ inspite of being killed in $n_4$?
- Identify the instance of dead code elimination $z = x$ in $n_6$
- Would the first round of dead code elimination cause liveness information to change? Yes
- Would the second round of liveness analysis lead to further dead code elimination?
Interpreting the Result of Liveness Analysis for Tutorial Problem 3

- Why is \( z \) live at the exit of \( n_5 \)?
- Why is \( z \) not live at the entry of \( n_5 \)?
- Why is \( x \) live at the exit of \( n_3 \) inspite of being killed in \( n_4 \)?
- Identify the instance of dead code elimination \( z = x \) in \( n_6 \)
- Would the first round of dead code elimination cause liveness information to change? Yes
- Would the second round of liveness analysis lead to further dead code elimination? Yes
Choice of Initialization

What should be the initial value of internal nodes?

The role of boundary info $B_I$ explained later in the context of available expressions analysis.
Choice of Initialization

What should be the initial value of internal nodes?

- Confluence is $\cup$
- Identity of $\cup$ is $\emptyset$

The role of boundary info $B_l$ explained later in the context of available expressions analysis
Choice of Initialization

What should be the initial value of internal nodes?

- Confluence is $\cup$
- Identity of $\cup$ is $\emptyset$
- We begin with $\emptyset$ and let the sets at each program point grow

A revisit to a program point

- may consider a new execution path
- more variables may be found to be live
- a variable found to be live earlier does not become dead

The role of boundary info $BI$ explained later in the context of available expressions analysis
How Does the Initialization Affect the Solution?

$a = b = 5$

print $b$
How Does the Initialization Affect the Solution?

\[ a = b = 5 \]

\[ \text{print } b \]
How Does the Initialization Affect the Solution?

Init. | Iter.
---|----
\(a = b = 5\) | #1
\(\emptyset\) | \(\emptyset\)
\(\emptyset\) | \(\emptyset\)
\(\emptyset\) | \(\emptyset\)
\(\emptyset\) | \(\emptyset\)
\(\emptyset\) | \(\emptyset\)
How Does the Initialization Affect the Solution?

\[ a = b = 5 \]

Init. | Iter.
--- | ---
∅ | #1
∅ | 
∅ | 
∅ | 
∅ | 
∅ | 
∅ | 
∅ | 
∅ | 
∅ | ∅

Jul 2017 IIT Bombay
How Does the Initialization Affect the Solution?

\[ a = b = 5 \]

\[
\begin{array}{c}
\text{Init.} \\
\emptyset \\
\emptyset \\
\emptyset \\
\emptyset \\
\emptyset \\
\emptyset \\
\emptyset \\
\end{array}
\]

\[
\begin{array}{c}
\text{Iter.} \\
\emptyset \\
\emptyset \\
\emptyset \\
\emptyset \\
\end{array}
\]
How Does the Initialization Affect the Solution?

\[ a = b = 5 \]

Init. | Iter. #1
--- | ---
\emptyset | \emptyset
\emptyset | \{b\}
\emptyset | \emptyset
\emptyset | \emptyset
\emptyset | \emptyset

Jul 2017 IIT Bombay
How Does the Initialization Affect the Solution?

```
a = b = 5
print b
```

<table>
<thead>
<tr>
<th>Init.</th>
<th>Iter. #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>∅</td>
<td>{b}</td>
</tr>
<tr>
<td>∅</td>
<td>{b}</td>
</tr>
<tr>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>
How Does the Initialization Affect the Solution?

\[ a = b = 5 \]

<table>
<thead>
<tr>
<th>Init.</th>
<th>Iter. #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>∅</td>
<td>{b}</td>
</tr>
<tr>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>

Jul 2017 IIT Bombay
How Does the Initialization Affect the Solution?

\[ a = b = 5 \]

<table>
<thead>
<tr>
<th>Init.</th>
<th>Iter. #1</th>
<th>Iter. #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( \emptyset )</td>
<td>{b}</td>
<td>{b}</td>
</tr>
<tr>
<td>( \emptyset )</td>
<td>{b}</td>
<td>{b}</td>
</tr>
<tr>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>{b}</td>
</tr>
<tr>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
How Does the Initialization Affect the Solution?

<table>
<thead>
<tr>
<th>Init.</th>
<th>Iter. #1</th>
<th>Iter. #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>∅</td>
<td>{b}</td>
<td>{b}</td>
</tr>
<tr>
<td>∅</td>
<td>∅</td>
<td>{b}</td>
</tr>
<tr>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Init.</th>
<th>Iter. #1</th>
<th>Iter. #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>{a, b}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{a, b}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{a, b}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{a, b}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Jul 2017

IIT Bombay
How Does the Initialization Affect the Solution?

\[ a = b = 5 \]

Print \( b \)

<table>
<thead>
<tr>
<th>Init.</th>
<th>Iter. #1</th>
<th>Iter. #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( \emptyset )</td>
<td>( { b } )</td>
<td>( { b } )</td>
</tr>
<tr>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( { b } )</td>
</tr>
</tbody>
</table>

\[ a = b = 5 \]

Print \( b \)

<table>
<thead>
<tr>
<th>Init.</th>
<th>Iter. #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( { a, b } )</td>
<td>( { a, b } )</td>
</tr>
<tr>
<td>( { a, b } )</td>
<td>( { a, b } )</td>
</tr>
<tr>
<td>( { a, b } )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>

Jul 2017 IIT Bombay
How Does the Initialization Affect the Solution?

<table>
<thead>
<tr>
<th>Init.</th>
<th>Iter. #1</th>
<th>Iter. #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$\emptyset$</td>
<td>${b}$</td>
<td>${b}$</td>
</tr>
<tr>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>${b}$</td>
</tr>
<tr>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

$a = b = 5$

Print $b$

<table>
<thead>
<tr>
<th>Init.</th>
<th>Iter. #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>${a, b}$</td>
<td>${a, b}$</td>
</tr>
<tr>
<td>${a, b}$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

$a = b = 5$

Print $b$
How Does the Initialization Affect the Solution?

\[ a = b = 5 \]

<table>
<thead>
<tr>
<th></th>
<th>Init.</th>
<th>Iter. #1</th>
<th>Iter. #2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>φ</td>
<td>φ</td>
<td>φ</td>
</tr>
<tr>
<td></td>
<td>φ</td>
<td>{b}</td>
<td>{b}</td>
</tr>
<tr>
<td>print b</td>
<td>φ</td>
<td>φ</td>
<td>{b}</td>
</tr>
<tr>
<td></td>
<td>φ</td>
<td>φ</td>
<td>φ</td>
</tr>
<tr>
<td></td>
<td>φ</td>
<td>φ</td>
<td>φ</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Init.</th>
<th>Iter. #1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>{a, b}</td>
<td>{a, b}</td>
</tr>
<tr>
<td>print b</td>
<td>{a, b}</td>
<td>{a, b}</td>
</tr>
<tr>
<td></td>
<td>{a, b}</td>
<td>{a, b}</td>
</tr>
<tr>
<td></td>
<td>{a, b}</td>
<td>φ</td>
</tr>
<tr>
<td></td>
<td>φ</td>
<td>φ</td>
</tr>
</tbody>
</table>
How Does the Initialization Affect the Solution?

\begin{align*}
\text{Init.} & \quad \text{Iter.} \ #1 & \quad \text{Iter.} \ #2 \\
\emptyset & \quad \emptyset & \quad \emptyset \\
\emptyset & \quad \{b\} & \quad \{b\} \\
\emptyset & \quad \emptyset & \quad \{b\} \\
\emptyset & \quad \emptyset & \quad \emptyset \\
\emptyset & \quad \{b\} & \quad \{b\} \\
\emptyset & \quad \emptyset & \quad \emptyset \\
\emptyset & \quad \{b\} & \quad \{b\} \\
\end{align*}
How Does the Initialization Affect the Solution?

\[
a = b = 5
\]

<table>
<thead>
<tr>
<th>Init.</th>
<th>Iter. #1</th>
<th>Iter. #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>∅</td>
<td>{b}</td>
<td>{b}</td>
</tr>
<tr>
<td>∅</td>
<td>∅</td>
<td>{b}</td>
</tr>
<tr>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>

Jul 2017 IIT Bombay
How Does the Initialization Affect the Solution?

\[
a = b = 5
\]

<table>
<thead>
<tr>
<th>Init.</th>
<th>Iter. #1</th>
<th>Iter. #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>\emptyset</td>
<td>\emptyset</td>
<td>\emptyset</td>
</tr>
<tr>
<td>\emptyset</td>
<td>{b}</td>
<td>{b}</td>
</tr>
<tr>
<td>\emptyset</td>
<td>\emptyset</td>
<td>{b}</td>
</tr>
<tr>
<td>\emptyset</td>
<td>\emptyset</td>
<td>\emptyset</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Init.</th>
<th>Iter. #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{a, b}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>{a, b}</td>
<td>{a, b}</td>
</tr>
<tr>
<td>{a, b}</td>
<td>{a, b}</td>
</tr>
<tr>
<td>{a, b}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>\emptyset</td>
<td>\emptyset</td>
</tr>
</tbody>
</table>

Jul 2017
IIT Bombay
How Does the Initialization Affect the Solution?

<table>
<thead>
<tr>
<th>Init.</th>
<th>Iter. #1</th>
<th>Iter. #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>∅</td>
<td>{b}</td>
<td>{b}</td>
</tr>
<tr>
<td>∅</td>
<td>∅</td>
<td>{b}</td>
</tr>
<tr>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>

\[ a = b = 5 \]

\[ \text{print } b \]

\[ a \text{ is spuriously marked live} \]
Soundness and Precision of Live Variables Analysis

Consider dead code elimination based on liveness information
Soundness and Precision of Live Variables Analysis

Consider dead code elimination based on liveness information

- Spurious inclusion of a non-live variable

\[
\begin{align*}
  x &= y + 10 \\
  \text{Out}_i &= \{x, y\} \\
  \text{print } y \\
  \text{End}
\end{align*}
\]
Soundness and Precision of Live Variables Analysis

Consider dead code elimination based on liveness information

- Spurious inclusion of a non-live variable
  - A dead assignment may not be eliminated
  - Solution is sound but may be imprecise

```
x = y + 10
```

```
print y
```

```
End
```

```
Out_i = \{x, y\}
```

```
i
j
```
Soundness and Precision of Live Variables Analysis

Consider dead code elimination based on liveness information

- Spurious inclusion of a non-live variable
  - A dead assignment may not be eliminated
  - Solution is sound but may be imprecise

- Spurious exclusion of a live variable

$x = y + 10$

$Out_i = \{x, y\}$

print $y$

End

$x = z + 10$

$Out_i = \{y\}$

print $x, y$

End
Soundness and Precision of Live Variables Analysis

Consider dead code elimination based on liveness information

- **Spurious inclusion of a non-live variable**
  - A dead assignment may not be eliminated
  - Solution is sound but may be imprecise

- **Spurious exclusion of a live variable**
  - A useful assignment may be eliminated
  - Solution is unsound

```plaintext
x = y + 10
print y
End
```

```
x = z + 10
print x, y
End
```
Consider dead code elimination based on liveness information

- Spurious inclusion of a non-live variable
  - A dead assignment may not be eliminated
  - Solution is sound but may be imprecise

- Spurious exclusion of a live variable
  - A useful assignment may be eliminated
  - Solution is unsound

- Given $L_2 \supseteq L_1$ representing liveness information
  - Using $L_2$ in place of $L_1$ is sound
  - Using $L_1$ in place of $L_2$ may not be sound

```
x = y + 10
print y
print x
```

```
x = z + 10
print x, y
```
Soundness and Precision of Live Variables Analysis

Consider dead code elimination based on liveness information

- Spurious inclusion of a non-live variable
  - A dead assignment may not be eliminated
  - Solution is sound but may be imprecise

- Spurious exclusion of a live variable
  - A useful assignment may be eliminated
  - Solution is unsound

- Given $L_2 \supseteq L_1$ representing liveness information
  - Using $L_2$ in place of $L_1$ is sound
  - Using $L_1$ in place of $L_2$ may not be sound

- The smallest set of all live variables is most precise
  - Since liveness sets grow (confluence is $\cup$), we choose $\emptyset$ as the initial conservative value
Termination, Convergence, and Complexity

• For live variables analysis,
  ▶ The set of all variables is finite, and
  ▶ the confluence operation (i.e. meet) is union, hence
  ▶ the set associated with a data flow variable can only grow

⇒ Termination is guaranteed
Termination, Convergence, and Complexity

- For live variables analysis,
  - The set of all variables is finite, and
  - the confluence operation (i.e. meet) is union, hence
  - the set associated with a data flow variable can only grow

  \[\Rightarrow\] Termination is guaranteed

- Since initial value is \(\emptyset\), live variables analysis converges on the smallest set
Termination, Convergence, and Complexity

• For live variables analysis,
  ▶ The set of all variables is finite, and
  ▶ the confluence operation (i.e. meet) is union, hence
  ▶ the set associated with a data flow variable can only grow

⇒ Termination is guaranteed

• Since initial value is $\emptyset$, live variables analysis converges on the smallest set

• How many iterations do we need for reaching the convergence?
Termination, Convergence, and Complexity

• For live variables analysis,
  ▶ The set of all variables is finite, and
  ▶ the confluence operation (i.e. meet) is union, hence
  ▶ the set associated with a data flow variable can only grow

⇒ Termination is guaranteed

• Since initial value is $\emptyset$, live variables analysis converges on the smallest set

• How many iterations do we need for reaching the convergence?

• Going beyond live variables analysis
  ▶ Do the sets always grow for other data flow frameworks?
  ▶ What is the complexity of round robin analysis for other analyses?
Termination, Convergence, and Complexity

- For live variables analysis,
  - The set of all variables is finite, and
  - the confluence operation (i.e. meet) is union, hence
  - the set associated with a data flow variable can only grow

  \[\Rightarrow\] Termination is guaranteed

- Since initial value is \(\emptyset\), live variables analysis converges on the smallest set

- How many iterations do we need for reaching the convergence?

- Going beyond live variables analysis
  - Do the sets always grow for other data flow frameworks?
  - What is the complexity of round robin analysis for other analyses?

Answered formally in module 2 (Theoretical Abstractions)
Conservative Nature of Analysis (1)

\[ x = \text{abs}(x) \]

\[ \text{if } (x < 0) \]

\[ x = a + y \]
\[ x = a + z \]

\[ \text{T} \]
\[ \text{F} \]
Conservative Nature of Analysis (1)

- $\text{abs}(n)$ returns the absolute value of $n$
Conservative Nature of Analysis (1)

- abs(n) returns the absolute value of n
- Is y live on entry to block b2?
Conservative Nature of Analysis (1)

- $\text{abs}(n)$ returns the absolute value of $n$
- Is $y$ live on entry to block $b2$?
- By execution semantics, NO
  Path $b1 \rightarrow b2 \rightarrow b3$ is an infeasible execution path

\[
x = \text{abs}(x)
\]

\[
\text{if} \ (x < 0)
\]

\[
x = a + y
\]

\[
x = a + z
\]
Conservative Nature of Analysis (1)

- \( \text{abs}(n) \) returns the absolute value of \( n \)
- Is \( y \) live on entry to block \( b2 \)?
- By execution semantics, NO
  Path \( b1 \rightarrow b2 \rightarrow b3 \) is an infeasible execution path
- A compiler makes conservative assumptions:
  \textit{All branch outcomes are possible}
  \( \Rightarrow \) Consider every path in CFG as a potential execution path
Conservative Nature of Analysis (1)

- abs(n) returns the absolute value of n
- Is y live on entry to block b2?
- By execution semantics, NO
  Path b1→b2→b3 is an infeasible execution path
- A compiler makes conservative assumptions:
  *All branch outcomes are possible*
  ⇒ Consider every path in CFG as a potential execution path
- Our analysis concludes that y is live on entry to block b2
Conservative Nature of Analysis (2)

```
if (x < 0) b1
a = a + y b2
if (x < 0) b4
x = a + z b3
x = c + 1 b5
x = b + 1 b6
b7
```
Conservative Nature of Analysis (2)

- Is b live on entry to block b2?
Conservative Nature of Analysis (2)

- Is b live on entry to block b2?
- By execution semantics, NO
  Path $b_1 \rightarrow b_2 \rightarrow b_4 \rightarrow b_6$ is an infeasible execution path
Conservative Nature of Analysis (2)

- Is b live on entry to block b2?
  - By execution semantics, NO
  - Path $b_1 \rightarrow b_2 \rightarrow b_4 \rightarrow b_6$ is an infeasible execution path

- Is c live on entry to block b3?
  - Path $b_1 \rightarrow b_3 \rightarrow b_4 \rightarrow b_6$ is a feasible execution path
Conservative Nature of Analysis (2)

- Is \( b \) live on entry to block \( b_2 \)?
  - By execution semantics, NO
  - Path \( b_1 \rightarrow b_2 \rightarrow b_4 \rightarrow b_6 \) is an infeasible execution path

- Is \( c \) live on entry to block \( b_3 \)?
  - Path \( b_1 \rightarrow b_3 \rightarrow b_4 \rightarrow b_6 \) is a feasible execution path

- A compiler makes conservative assumptions
  \( \Rightarrow \) our analysis is *path insensitive*

Note: It is *flow sensitive* (i.e. information is computed for every control flow points)
Conservative Nature of Analysis (2)

- Is $b$ live on entry to block $b2$?
- By execution semantics, NO
  Path $b1 \rightarrow b2 \rightarrow b4 \rightarrow b6$ is an infeasible execution path

- Is $c$ live on entry to block $b3$?
  Path $b1 \rightarrow b3 \rightarrow b4 \rightarrow b6$ is a feasible execution path

- A compiler make conservative assumptions $\Rightarrow$ our analysis is *path insensitive*

  Note: It is *flow sensitive* (i.e. information is computed for every control flow points)

- Our analysis concludes that $b$ is live at the entry of $b2$
Conservative Nature of Analysis (2)

- Is $b$ live on entry to block $b2$?
- By execution semantics, NO
  Path $b1 \rightarrow b2 \rightarrow b4 \rightarrow b6$ is an infeasible execution path

- Is $c$ live on entry to block $b3$?
  Path $b1 \rightarrow b3 \rightarrow b4 \rightarrow b6$ is a feasible execution path

- A compiler makes conservative assumptions $\Rightarrow$ our analysis is path insensitive
  Note: It is flow sensitive (i.e. information is computed for every control flow point)

- Our analysis concludes that $b$ is live at the entry of $b2$

- Is $c$ live at the entry of $b3$?
Conservative Nature of Analysis at Intraprocedural Level

- We assume that all paths are potentially executable
- Our analysis is path insensitive
  - The data flow information at a program point \( p \) is path insensitive
    - information at \( p \) is merged along all paths reaching \( p \)
  - The data flow information reaching \( p \) is computed path insensitively
    - information is merged at all shared points in paths reaching \( p \)
    - may generate spurious information due to non-distributive flow functions

More about it in module 2
Conservative Nature of Analysis at Interprocedural Level

- Context insensitivity
  - Merges of information across all calling contexts
- Flow insensitivity
  - Disregards the control flow

More about it in module 4
What About Soundness of Analysis Results?

- No compromises
- We will study it in module 2
Part 4

Available Expressions Analysis
Defining Available Expressions Analysis

An expression $e$ is available at a program point $p$, if every path from program entry to $p$ contains an evaluation of $e$ which is not followed by a definition of any operand of $e$. 
Defining Available Expressions Analysis

An expression $e$ is available at a program point $p$, if every path from program entry to $p$ contains an evaluation of $e$ which is not followed by a definition of any operand of $e$. 

\[ a * b \text{ is available at } p \]
Defining Available Expressions Analysis

An expression \( e \) is available at a program point \( p \), if every path from program entry to \( p \) contains an evaluation of \( e \) which is not followed by a definition of any operand of \( e \).
Defining Available Expressions Analysis

An expression $e$ is available at a program point $p$, if every path from program entry to $p$ contains an evaluation of $e$ which is not followed by a definition of any operand of $e$. 

- $a \times b$ is available at $p$
- $a \times b$ is not available at $p$
- $a \times b$ is not available at $p$
Local Data Flow Properties for Available Expressions Analysis

\[ \text{Gen}_n = \{ e \mid \text{expression } e \text{ is evaluated in basic block } n \text{ and this evaluation is not followed by a definition of any operand of } e \} \]

\[ \text{Kill}_n = \{ e \mid \text{basic block } n \text{ contains a definition of an operand of } e \} \]

<table>
<thead>
<tr>
<th>Entity Manipulation</th>
<th>Exposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{Gen}_n</td>
<td>Expression</td>
</tr>
<tr>
<td>\text{Kill}_n</td>
<td>Expression</td>
</tr>
</tbody>
</table>
Data Flow Equations For Available Expressions Analysis

\[ \begin{align*}
\text{In}_n &= \begin{cases} 
  BI & \text{if } n \text{ is Start block} \\
  \bigcap_{p \in \text{pred}(n)} \text{Out}_p & \text{otherwise}
\end{cases} \\
\text{Out}_n &= \text{Gen}_{n} \cup (\text{In}_n - \text{Kill}_n)
\end{align*} \]
Data Flow Equations For Available Expressions Analysis

\[\begin{align*}
\text{In}_n &= \begin{cases} 
    BL & \text{if } n \text{ is Start block} \\
    \bigcap_{p \in \text{pred}(n)} \text{Out}_p & \text{otherwise}
\end{cases} \\
\text{Out}_n &= \text{Gen}_n \cup (\text{In}_n - \text{Kill}_n)
\end{align*}\]

Alternatively,

\[\begin{align*}
\text{Out}_n &= f_n(\text{In}_n), \quad \text{where} \\
f_n(X) &= \text{Gen}_n \cup (X - \text{Kill}_n)
\end{align*}\]
Data Flow Equations For Available Expressions Analysis

\[
\begin{align*}
In_n &= \begin{cases} 
Bl & \text{if } n \text{ is Start block} \\
\bigcap_{p \in \text{pred}(n)} Out_p & \text{otherwise}
\end{cases} \\
Out_n &= Gen_n \cup (In_n - Kill_n)
\end{align*}
\]

Alternatively,
\[
Out_n = f_n(In_n), \quad \text{where}
\]
\[
f_n(X) = Gen_n \cup (X - Kill_n)
\]

- \(In_n\) and \(Out_n\) are sets of expressions
Data Flow Equations For Available Expressions Analysis

\[ In_n = \begin{cases} B & \text{if } n \text{ is Start block} \\ \bigcap_{p \in \text{pred}(n)} Out_p & \text{otherwise} \end{cases} \]

\[ Out_n = Gen_n \cup (In_n - Kill_n) \]

Alternatively,
\[ Out_n = f_n(In_n), \quad \text{where} \]
\[ f_n(X) = Gen_n \cup (X - Kill_n) \]

- \( In_n \) and \( Out_n \) are sets of expressions
- \( BI \) is \( \emptyset \) for expressions involving a local variable
Using Data Flow Information of Available Expressions Analysis

- Common subexpression elimination
Using Data Flow Information of Available Expressions Analysis

- Common subexpression elimination
  - If an expression is available at the entry of a block $n$ ($\ln n$) and
Using Data Flow Information of Available Expressions Analysis

• Common subexpression elimination
  ▶ If an expression is available at the entry of a block \( n \) (\( In_n \)) and
  ▶ a computation of the expression exists in \( n \) such that
Using Data Flow Information of Available Expressions Analysis

- Common subexpression elimination
  - If an expression is available at the entry of a block $n$ ($In_n$) and
  - a computation of the expression exists in $n$ such that
  - it is not preceded by a definition of any of its operands ($AntGen_n$)
Using Data Flow Information of Available Expressions Analysis

- Common subexpression elimination
  - If an expression is available at the entry of a block \( n \) \((\text{In}_n)\) and
  - a computation of the expression exists in \( n \) such that
  - it is not preceded by a definition of any of its operands \((\text{AntGen}_n)\)

Then the expression is redundant

\[
\text{Redundant}_n = \text{In}_n \cap \text{AntGen}_n
\]
Using Data Flow Information of Available Expressions Analysis

- Common subexpression elimination
  - If an expression is available at the entry of a block $n$ ($In_n$) and
  - a computation of the expression exists in $n$ such that
  - it is not preceded by a definition of any of its operands ($AntGen_n$)

  Then the expression is redundant

  $$\text{Redundant}_n = In_n \cap AntGen_n$$

- A redundant expression is upwards exposed whereas the expressions in $Gen_n$ are downwards exposed
An Example of Available Expressions Analysis

Let $e_1 \equiv a \ast b$, $e_2 \equiv b \ast c$, $e_3 \equiv c \ast d$, $e_4 \equiv d \ast e$

<table>
<thead>
<tr>
<th>Node</th>
<th>Gen</th>
<th>Kill</th>
<th>Available</th>
<th>Redund.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>${e_1, e_2}$</td>
<td>1100</td>
<td>$\emptyset$</td>
<td>0000</td>
</tr>
<tr>
<td>2</td>
<td>${e_3}$</td>
<td>0010</td>
<td>$\emptyset$</td>
<td>0000</td>
</tr>
<tr>
<td>3</td>
<td>$\emptyset$</td>
<td>0000</td>
<td>${e_2, e_3}$</td>
<td>0110</td>
</tr>
<tr>
<td>4</td>
<td>$\emptyset$</td>
<td>0000</td>
<td>${e_3, e_4}$</td>
<td>0011</td>
</tr>
<tr>
<td>5</td>
<td>${e_1, e_4}$</td>
<td>1001</td>
<td>$\emptyset$</td>
<td>0000</td>
</tr>
<tr>
<td>6</td>
<td>${e_4}$</td>
<td>0001</td>
<td>$\emptyset$</td>
<td>0000</td>
</tr>
</tbody>
</table>
An Example of Available Expressions Analysis

Let $e_1 \equiv a \times b$, $e_2 \equiv b \times c$, $e_3 \equiv c \times d$, $e_4 \equiv d \times e$

### Initialisation

```
0000  \\
1111  \\
1111  \\
1111  \\
1111  \\
1111  \\
```

```
a \times b  \\
b \times c  \\
c \times d  \\
c = 2  \\
d = 3  \\
d \times e  \\
a \times b  \\
d \times e  \\
```

### Table

<table>
<thead>
<tr>
<th>Node</th>
<th>Gen</th>
<th>Kill</th>
<th>Available</th>
<th>Redund.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>${e_1, e_2}$</td>
<td>1100</td>
<td>$\emptyset$</td>
<td>0000</td>
</tr>
<tr>
<td>2</td>
<td>${e_3}$</td>
<td>0010</td>
<td>$\emptyset$</td>
<td>0000</td>
</tr>
<tr>
<td>3</td>
<td>$\emptyset$</td>
<td>0000</td>
<td>${e_2, e_3}$</td>
<td>0110</td>
</tr>
<tr>
<td>4</td>
<td>$\emptyset$</td>
<td>0000</td>
<td>${e_3, e_4}$</td>
<td>0011</td>
</tr>
<tr>
<td>5</td>
<td>${e_1, e_4}$</td>
<td>1001</td>
<td>$\emptyset$</td>
<td>0000</td>
</tr>
<tr>
<td>6</td>
<td>${e_4}$</td>
<td>0001</td>
<td>$\emptyset$</td>
<td>0000</td>
</tr>
</tbody>
</table>
An Example of Available Expressions Analysis

Let $e_1 \equiv a \times b$, $e_2 \equiv b \times c$, $e_3 \equiv c \times d$, $e_4 \equiv d \times e$

<table>
<thead>
<tr>
<th>Node</th>
<th>Gen</th>
<th>Kill</th>
<th>Available</th>
<th>Redund.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>${e_1, e_2}$</td>
<td>1100</td>
<td>$\emptyset$</td>
<td>0000</td>
</tr>
<tr>
<td>2</td>
<td>${e_3}$</td>
<td>0010</td>
<td>$\emptyset$</td>
<td>0000</td>
</tr>
<tr>
<td>3</td>
<td>$\emptyset$</td>
<td>0000</td>
<td>${e_2, e_3}$</td>
<td>0110</td>
</tr>
<tr>
<td>4</td>
<td>$\emptyset$</td>
<td>0000</td>
<td>${e_3, e_4}$</td>
<td>0011</td>
</tr>
<tr>
<td>5</td>
<td>${e_1, e_4}$</td>
<td>1001</td>
<td>$\emptyset$</td>
<td>0000</td>
</tr>
<tr>
<td>6</td>
<td>${e_4}$</td>
<td>0001</td>
<td>$\emptyset$</td>
<td>0000</td>
</tr>
</tbody>
</table>
An Example of Available Expressions Analysis

Let \( e_1 \equiv a \times b \), \( e_2 \equiv b \times c \), \( e_3 \equiv c \times d \), \( e_4 \equiv d \times e \)

### Iteration #2

<table>
<thead>
<tr>
<th>Node</th>
<th>( \text{Gen} )</th>
<th>( \text{Kill} )</th>
<th>( \text{Available} )</th>
<th>( \text{Redund.} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{( e_1, e_2 )}</td>
<td>1100</td>
<td>( \emptyset )</td>
<td>0000</td>
</tr>
<tr>
<td>2</td>
<td>{( e_3 )}</td>
<td>0010</td>
<td>( \emptyset )</td>
<td>0000</td>
</tr>
<tr>
<td>3</td>
<td>( \emptyset )</td>
<td>0000</td>
<td>{( e_2, e_3 )}</td>
<td>0110</td>
</tr>
<tr>
<td>4</td>
<td>( \emptyset )</td>
<td>0000</td>
<td>{( e_3, e_4 )}</td>
<td>0011</td>
</tr>
<tr>
<td>5</td>
<td>{( e_1, e_4 )}</td>
<td>1001</td>
<td>( \emptyset )</td>
<td>0000</td>
</tr>
<tr>
<td>6</td>
<td>{( e_4 )}</td>
<td>0001</td>
<td>( \emptyset )</td>
<td>0000</td>
</tr>
</tbody>
</table>
An Example of Available Expressions Analysis

Let \( e_1 \equiv a \times b, e_2 \equiv b \times c, e_3 \equiv c \times d, e_4 \equiv d \times e \)

<table>
<thead>
<tr>
<th>Node</th>
<th>Gen</th>
<th>Kill</th>
<th>Available</th>
<th>Redund.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( {e_1, e_2} ) 1100</td>
<td>( \emptyset )</td>
<td>0000</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>2</td>
<td>( {e_3} ) 0010</td>
<td>( \emptyset )</td>
<td>0000</td>
<td>( {e_1} )</td>
</tr>
<tr>
<td>3</td>
<td>( \emptyset ) 0000</td>
<td>( {e_2, e_3} ) 0110</td>
<td>( {e_1, e_3} )</td>
<td>1010</td>
</tr>
<tr>
<td>4</td>
<td>( \emptyset ) 0000</td>
<td>( {e_3, e_4} ) 0011</td>
<td>( {e_1, e_3} )</td>
<td>1010</td>
</tr>
<tr>
<td>5</td>
<td>( {e_1, e_4} ) 1001</td>
<td>( \emptyset )</td>
<td>0000</td>
<td>( {e_1} )</td>
</tr>
<tr>
<td>6</td>
<td>( {e_4} ) 0001</td>
<td>( \emptyset )</td>
<td>0000</td>
<td>( {e_1, e_4} )</td>
</tr>
</tbody>
</table>
Tutorial Problem 2 for Available Expressions Analysis

\[ n_1 \]
\[ d = a \times b \]
\[ e = b + c \]

\[ n_2 \]
\[ if (c) \]

\[ n_3 \]
\[ a = b + c \]

\[ n_4 \]
\[ c = a \times b \]
\[ a = 10 \]

\[ n_5 \]
\[ if (d) \]

\[ n_6 \]
\[ print \ a, b, c, d \]

\[ \text{Expr} = \{ a \times b, b + c \} \]
### Solution of the Tutorial Problem 2

Bit vector $a \times b \overset{\text{Global Information}}{\rightarrow} b + c$

<table>
<thead>
<tr>
<th>Node</th>
<th>Local Information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$Gen_n$</td>
<td>$Kill_n$</td>
</tr>
<tr>
<td>$n_1$</td>
<td>11</td>
<td>00</td>
</tr>
<tr>
<td>$n_2$</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>$n_3$</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>$n_4$</td>
<td>00</td>
<td>11</td>
</tr>
<tr>
<td>$n_5$</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>$n_6$</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>
Solution of the Tutorial Problem 2

Bit vector $a \ast b \mid b + c$

<table>
<thead>
<tr>
<th>Node</th>
<th>Local Information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$Gen_n$</td>
<td>$Kill_n$</td>
</tr>
<tr>
<td>$n_1$</td>
<td>11</td>
<td>00</td>
</tr>
<tr>
<td>$n_2$</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>$n_3$</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>$n_4$</td>
<td>00</td>
<td>11</td>
</tr>
<tr>
<td>$n_5$</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>$n_6$</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>

$Redundant_n$
Solution of the Tutorial Problem 2

Bit vector $a * b | b + c$

<table>
<thead>
<tr>
<th>Node</th>
<th>Local Information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$Gen_n$ $Kill_n$ $AntGen_n$</td>
<td>$In_n$ $Out_n$ $In_n$ $Out_n$</td>
</tr>
<tr>
<td>$n_1$</td>
<td>11 00 11</td>
<td>00 11</td>
</tr>
<tr>
<td>$n_2$</td>
<td>00 00 00</td>
<td>11 11 00 00</td>
</tr>
<tr>
<td>$n_3$</td>
<td>01 10 01</td>
<td>11 01 00</td>
</tr>
<tr>
<td>$n_4$</td>
<td>00 11 10</td>
<td>11 00 00</td>
</tr>
<tr>
<td>$n_5$</td>
<td>00 00 00</td>
<td>00 00</td>
</tr>
<tr>
<td>$n_6$</td>
<td>00 00 00</td>
<td>00 00</td>
</tr>
</tbody>
</table>
Solution of the Tutorial Problem 2

Bit vector $a \ast b \mid b + c$

<table>
<thead>
<tr>
<th>Node</th>
<th>Local Information</th>
<th>Global Information</th>
<th>Global Information</th>
<th>Redundant$_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gen$_n$</td>
<td>Kill$_n$</td>
<td>AntGen$_n$</td>
<td>Iteration # 1</td>
</tr>
<tr>
<td>$n_1$</td>
<td>11</td>
<td>00</td>
<td>11</td>
<td>00</td>
</tr>
<tr>
<td>$n_2$</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>11</td>
</tr>
<tr>
<td>$n_3$</td>
<td>01</td>
<td>10</td>
<td>01</td>
<td>11</td>
</tr>
<tr>
<td>$n_4$</td>
<td>00</td>
<td>11</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>$n_5$</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>$n_6$</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>
Tutorial Problem 3 for Available Expressions Analysis

$\begin{align*}
    n_1: & \quad c = a \ast b \\
      & \quad d = b + c \\
    n_2: & \quad d = a + b \\
    n_3: & \quad d = b + c \\
    n_4: & \quad a = 5 \\
      & \quad d = a + b \\
    n_5: & \quad c = 10 \\
    n_6: & \quad d = a + b \\
      & \quad \text{print } a, b, c, d
\end{align*}$

$\mathbb{E}xpr = \{ a \ast b, b + c, a + b \}$
Solution of the Tutorial Problem 3

Bit vector $a \times b \quad b + c \quad a + b$

<table>
<thead>
<tr>
<th>Node</th>
<th>Local Information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gen$_n$</td>
<td>Kill$_n$</td>
</tr>
<tr>
<td>$n_1$</td>
<td>110</td>
<td>010</td>
</tr>
<tr>
<td>$n_2$</td>
<td>001</td>
<td>000</td>
</tr>
<tr>
<td>$n_3$</td>
<td>010</td>
<td>000</td>
</tr>
<tr>
<td>$n_4$</td>
<td>001</td>
<td>101</td>
</tr>
<tr>
<td>$n_5$</td>
<td>000</td>
<td>010</td>
</tr>
<tr>
<td>$n_6$</td>
<td>001</td>
<td>000</td>
</tr>
</tbody>
</table>
# Solution of the Tutorial Problem 3

Bit vector $a \times b \quad b + c \quad a + b$

<table>
<thead>
<tr>
<th>Node</th>
<th>Local Information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$Gen_n$ $Kill_n$ $AntGen_n$</td>
<td>$In_n$ $Out_n$ $In_n$ $Out_n$ $In_n$ $Out_n$ $Redundant_n$</td>
</tr>
<tr>
<td>$n_1$</td>
<td>110 010 100</td>
<td>000 110</td>
</tr>
<tr>
<td>$n_2$</td>
<td>001 000 001</td>
<td>110 111</td>
</tr>
<tr>
<td>$n_3$</td>
<td>010 000 010</td>
<td>111 111</td>
</tr>
<tr>
<td>$n_4$</td>
<td>001 101 000</td>
<td>111 011</td>
</tr>
<tr>
<td>$n_5$</td>
<td>000 010 000</td>
<td>111 101</td>
</tr>
<tr>
<td>$n_6$</td>
<td>001 000 001</td>
<td>101 101</td>
</tr>
</tbody>
</table>
Solution of the Tutorial Problem 3

Bit vector \( a \times b \) \( b + c \) \( a + b \)

<table>
<thead>
<tr>
<th>Node</th>
<th>Local Information</th>
<th>Global Information</th>
<th>Redundant (_n)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( Gen_n )</td>
<td>( Kill_n )</td>
<td>( AntGen_n )</td>
</tr>
<tr>
<td>( n_1 )</td>
<td>110</td>
<td>010</td>
<td>100</td>
</tr>
<tr>
<td>( n_2 )</td>
<td>001</td>
<td>000</td>
<td>001</td>
</tr>
<tr>
<td>( n_3 )</td>
<td>010</td>
<td>000</td>
<td>010</td>
</tr>
<tr>
<td>( n_4 )</td>
<td>001</td>
<td>101</td>
<td>000</td>
</tr>
<tr>
<td>( n_5 )</td>
<td>000</td>
<td>010</td>
<td>000</td>
</tr>
<tr>
<td>( n_6 )</td>
<td>001</td>
<td>000</td>
<td>001</td>
</tr>
</tbody>
</table>
Solution of the Tutorial Problem 3

Bit vector \( a \ast b \) \( b + c \) \( a + b \)

<table>
<thead>
<tr>
<th>Node</th>
<th>Local Information</th>
<th>Global Information</th>
<th>Global Information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( Gen_n )</td>
<td>( Kill_n )</td>
<td>( AntGen_n )</td>
<td>( In_n )</td>
</tr>
<tr>
<td>( n_1 )</td>
<td>110</td>
<td>010</td>
<td>100</td>
<td>000</td>
</tr>
<tr>
<td>( n_2 )</td>
<td>001</td>
<td>000</td>
<td>001</td>
<td>110</td>
</tr>
<tr>
<td>( n_3 )</td>
<td>010</td>
<td>000</td>
<td>010</td>
<td>111</td>
</tr>
<tr>
<td>( n_4 )</td>
<td>001</td>
<td>101</td>
<td>000</td>
<td>111</td>
</tr>
<tr>
<td>( n_5 )</td>
<td>000</td>
<td>010</td>
<td>000</td>
<td>111</td>
</tr>
<tr>
<td>( n_6 )</td>
<td>001</td>
<td>000</td>
<td>001</td>
<td>101</td>
</tr>
</tbody>
</table>
Solution of the Tutorial Problem 3

Bit vector $a \ast b \mid b + c \mid a + b$

<table>
<thead>
<tr>
<th>Node</th>
<th>Local Information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$Gen_n$</td>
<td>$Kill_n$</td>
</tr>
<tr>
<td>$n_1$</td>
<td>110</td>
<td>010</td>
</tr>
<tr>
<td>$n_2$</td>
<td>001</td>
<td>000</td>
</tr>
<tr>
<td>$n_3$</td>
<td>010</td>
<td>000</td>
</tr>
<tr>
<td>$n_4$</td>
<td>001</td>
<td>101</td>
</tr>
<tr>
<td>$n_5$</td>
<td>000</td>
<td>010</td>
</tr>
<tr>
<td>$n_6$</td>
<td>001</td>
<td>000</td>
</tr>
</tbody>
</table>
Solution of the Tutorial Problem 3

Bit vector $a \times b \mid b + c \mid a + b$

<table>
<thead>
<tr>
<th>Node</th>
<th>Local Information</th>
<th>Global Information</th>
<th>Redundant $n$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$Gen_n$ $Kill_n$ $AntGen_n$</td>
<td>$In_n$ $Out_n$</td>
<td>$In_n$ $Out_n$</td>
</tr>
<tr>
<td>$n_1$</td>
<td>110 010 100</td>
<td>000 110</td>
<td></td>
</tr>
<tr>
<td>$n_2$</td>
<td>001 000 001</td>
<td>110 111</td>
<td>100 101</td>
</tr>
<tr>
<td>$n_3$</td>
<td>010 000 010</td>
<td>111 111</td>
<td>001 011</td>
</tr>
<tr>
<td>$n_4$</td>
<td>001 101 000</td>
<td>111 011</td>
<td>011</td>
</tr>
<tr>
<td>$n_5$</td>
<td>000 010 000</td>
<td>111 101</td>
<td>001 001</td>
</tr>
<tr>
<td>$n_6$</td>
<td>001 000 001</td>
<td>101 101</td>
<td>001 001</td>
</tr>
</tbody>
</table>

Why do we need 3 iterations as against 2 for previous problems?
The Effect of $Bl$ and Initialization on a Solution

1. $w = a + c$

2. $x = a * c$

3. $y = a + c$
   $z = a * c$
The Effect of $BI$ and Initialization on a Solution

\[
w = a + c \\
x = a \ast c \\
y = a + c \\
z = a \ast c
\]

<table>
<thead>
<tr>
<th>$BI$</th>
<th>Node</th>
<th>Initialization $\mathbb{U}$</th>
<th>Initialization $\emptyset$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>1</td>
<td>$In_n$</td>
<td>$Out_n$</td>
</tr>
<tr>
<td>$\mathbb{U}$</td>
<td>2</td>
<td>$In_n$</td>
<td>$Out_n$</td>
</tr>
<tr>
<td>$\mathbb{U}$</td>
<td>3</td>
<td>$In_n$</td>
<td>$Out_n$</td>
</tr>
</tbody>
</table>
The Effect of $B/I$ and Initialization on a Solution

\[
\begin{align*}
1. & \quad w = a + c \\
2. & \quad x = a \ast c \\
3. & \quad y = a + c \\
   & \quad z = a \ast c
\end{align*}
\]

\begin{table}
\begin{tabular}{|c|c|c|c|
\hline
$B/I$ & Node & \text{Initialization $\mathbb{U}$} & \text{Initialization $\emptyset$} \\
& & $\text{In}_n$ & $\text{Out}_n$ & $\text{In}_n$ & $\text{Out}_n$ \\
\hline
\emptyset & 1 & 00 & 10 & & \\
& 2 & 10 & 11 & & \\
& 3 & 10 & 11 & & \\
\mathbb{U} & 1 & & & & \\
& 2 & & & & \\
& 3 & & & & \\
\hline
\end{tabular}
\end{table}
The Effect of $BI$ and Initialization on a Solution

**Bit Vector**

\[
\begin{array}{cc}
  a + c & a \ast c \\
\end{array}
\]

**$BI$ Node Initialization**

<table>
<thead>
<tr>
<th>$BI$</th>
<th>Node</th>
<th>Initialization $\cup$</th>
<th>Initialization $\emptyset$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$In_n$</td>
<td>$Out_n$</td>
<td>$In_n$</td>
</tr>
<tr>
<td>$\emptyset$</td>
<td>1</td>
<td>00</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>$\cup$</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Effect of $B/I$ and Initialization on a Solution

Bit Vector

\[
\begin{array}{cc}
\text{a + c} & \text{a * c} \\
\end{array}
\]

<table>
<thead>
<tr>
<th>$B/I$</th>
<th>Node</th>
<th>Initialization $\mathbb{U}$</th>
<th>Initialization $\emptyset$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$\text{In}_n$</td>
<td>$\text{Out}_n$</td>
</tr>
<tr>
<td>$\emptyset$</td>
<td>1</td>
<td>00</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>$\mathbb{U}$</td>
<td>1</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>
The Effect of \( BI \) and Initialization on a Solution

### Bit Vector

\[
\begin{array}{c|c|c}
  & a + c & a \times c \\
\end{array}
\]

<table>
<thead>
<tr>
<th>( BI )</th>
<th>Node</th>
<th>Initialization ( \cup )</th>
<th>Initialization ( \emptyset )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset )</td>
<td>1</td>
<td>00</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>( \cup )</td>
<td>1</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

1. \( w = a + c \)
2. \( x = a \times c \)
3. \( y = a + c \\
   z = a \times c \)
## The Effect of $BI$ and Initialization on a Solution

**Bit Vector**

\[
\begin{array}{|c|}
\hline
a + c & a \times c \\
\hline
\end{array}
\]

<table>
<thead>
<tr>
<th>$BI$</th>
<th>Node</th>
<th>Initialization $\cup$</th>
<th>Initialization $\emptyset$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$ln_n$</td>
<td>$Out_n$</td>
<td>$ln_n$</td>
</tr>
<tr>
<td>$\emptyset$</td>
<td>1</td>
<td>00</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>$\cup$</td>
<td>1</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

This represents the expected availability information leading to elimination of $a + c$ in node 3 ($a \times c$ is not redundant in node 3).
The Effect of $BI$ and Initialization on a Solution

Bit Vector

\[ a + c \quad a \ast c \]

<table>
<thead>
<tr>
<th>$BI$</th>
<th>Node</th>
<th>Initialization $\cup$</th>
<th>Initialization $\emptyset$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>1</td>
<td>00</td>
<td>10</td>
</tr>
<tr>
<td>$\emptyset$</td>
<td>2</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>$\emptyset$</td>
<td>3</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>$\cup$</td>
<td>1</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>$\cup$</td>
<td>2</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>$\cup$</td>
<td>3</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

This misses the availability of $a + c$ in node 3
The Effect of $BI$ and Initialization on a Solution

This makes $a \cdot c$ available in node 3 although its computation in node 3 is not redundant.

\[
\begin{align*}
1 & : w = a + c \\
2 & : x = a \cdot c \\
3 & : y = a + c, z = a \cdot c
\end{align*}
\]

<table>
<thead>
<tr>
<th>$BI$</th>
<th>Node</th>
<th>Initialization $\cup$</th>
<th>Initialization $\emptyset$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>1</td>
<td>00</td>
<td>10</td>
</tr>
<tr>
<td>$\emptyset$</td>
<td>2</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>$\emptyset$</td>
<td>3</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>$\cup$</td>
<td>1</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>$\cup$</td>
<td>2</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>$\cup$</td>
<td>3</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>
The Effect of $BI$ and Initialization on a Solution

This makes $a \times c$ available in node 3 and but misses the availability of $a + c$ in node 3.

<table>
<thead>
<tr>
<th>$BI$</th>
<th>Node</th>
<th>Initialization $\cup$</th>
<th>Initialization $\emptyset$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$In_n$</td>
<td>$Out_n$</td>
</tr>
<tr>
<td>$\emptyset$</td>
<td>1</td>
<td>00</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>$\cup$</td>
<td>1</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>
The Effect of $B\!I$ and Initialization on a Solution

Bit Vector

$$\begin{array}{c|c|c}
\text{Initialization } \cup & a + c & a \times c \\
\hline
\text{Initialization } \emptyset & & \\
\hline
\text{Node} & \text{In}_n & \text{Out}_n & \text{In}_n & \text{Out}_n \\
\hline
1 & & & & \\
2 & & & & \\
3 & & & & \\
\emptyset & & & & \\
2 & & & & \\
3 & & & & \\
\cup & & & & \\
2 & & & & \\
3 & & & & \\
\end{array}$$

1. $w = a + c$
2. $x = a \times c$
3. $y = a + c$
4. $z = a \times c$

Sound & Precise

Unsound
Some Observations

- Data flow equations do not require a particular order of computation
  
  - **Specification.** Data flow equations define what needs to be computed and not how it is to be computed
  - **Implementation.** Round robin iterations perform the actual computation
  - Specification and implementation are distinct

- Initialization governs the quality of solution found
  
  - Only precision is affected, soundness is guaranteed
  - Associated with “internal” nodes

- $B_I$ depends on the semantics of the calling context
  
  - May cause unsoundness
  - Associated with “boundary” node (specified by data flow equations)
  
  Does not vary with the method or order of traversal
A New Data Flow Framework: Partially available expressions analysis

- Expressions that are computed and remain unmodified along some path reaching $p$

- The data flow equations are same as that of available expressions analysis except that the confluence is changed to $\cup$

Perform partially available expressions analysis for the example program used for available expressions analysis
Solution of the Tutorial Problem 2 for Partial Availability Analysis

Bit vector $a \times b \vert b + c$

<table>
<thead>
<tr>
<th>Node</th>
<th>Local Information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$Gen_n$</td>
<td>$Kill_n$</td>
</tr>
<tr>
<td>$n_1$</td>
<td>11</td>
<td>00</td>
</tr>
<tr>
<td>$n_2$</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>$n_3$</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>$n_4$</td>
<td>00</td>
<td>11</td>
</tr>
<tr>
<td>$n_5$</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>$n_6$</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>
Solution of the Tutorial Problem 2 for Partial Availability Analysis

Bit vector $a \ast b \mid b + c$

<table>
<thead>
<tr>
<th>Node</th>
<th>Local Information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$Gen_n$ $Kill_n$ $AntGen_n$</td>
<td>$PavIn_n$ $PavOut_n$ $ParRedund_n$</td>
</tr>
<tr>
<td>$n_1$</td>
<td>11 00 11</td>
<td>00 11</td>
</tr>
<tr>
<td>$n_2$</td>
<td>00 00 00</td>
<td>11 11</td>
</tr>
<tr>
<td>$n_3$</td>
<td>01 10 01</td>
<td>11 01</td>
</tr>
<tr>
<td>$n_4$</td>
<td>00 11 10</td>
<td>11 00</td>
</tr>
<tr>
<td>$n_5$</td>
<td>00 00 00</td>
<td>01 01</td>
</tr>
<tr>
<td>$n_6$</td>
<td>00 00 00</td>
<td>01 01</td>
</tr>
</tbody>
</table>
Solution of the Tutorial Problem 2 for Partial Availability Analysis

Bit vector $a \times b | b + c$

<table>
<thead>
<tr>
<th>Node</th>
<th>Local Information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$Gen_n$</td>
<td>$Kill_n$</td>
</tr>
<tr>
<td>$n_1$</td>
<td>11</td>
<td>00</td>
</tr>
<tr>
<td>$n_2$</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>$n_3$</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>$n_4$</td>
<td>00</td>
<td>11</td>
</tr>
<tr>
<td>$n_5$</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>$n_6$</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>
## Solution of the Tutorial Problem 3 for Partial Availability Analysis

Bit vector: \( a \times b \quad b + c \quad a + b \)

<table>
<thead>
<tr>
<th>Node</th>
<th>Local Information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( Gen_n )</td>
<td>( Kill_n )</td>
</tr>
<tr>
<td>( n_1 )</td>
<td>110</td>
<td>010</td>
</tr>
<tr>
<td>( n_2 )</td>
<td>001</td>
<td>000</td>
</tr>
<tr>
<td>( n_3 )</td>
<td>010</td>
<td>000</td>
</tr>
<tr>
<td>( n_4 )</td>
<td>001</td>
<td>101</td>
</tr>
<tr>
<td>( n_5 )</td>
<td>000</td>
<td>010</td>
</tr>
<tr>
<td>( n_6 )</td>
<td>001</td>
<td>000</td>
</tr>
</tbody>
</table>
Solution of the Tutorial Problem 3 for Partial Availability Analysis

Bit vector \[ a \times b \mid b + c \mid a + b \]

<table>
<thead>
<tr>
<th>Node</th>
<th>Local Information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gen$_n$</td>
<td>Kill$_n$</td>
</tr>
<tr>
<td>$n_1$</td>
<td>110</td>
<td>010</td>
</tr>
<tr>
<td>$n_2$</td>
<td>001</td>
<td>000</td>
</tr>
<tr>
<td>$n_3$</td>
<td>010</td>
<td>000</td>
</tr>
<tr>
<td>$n_4$</td>
<td>001</td>
<td>101</td>
</tr>
<tr>
<td>$n_5$</td>
<td>000</td>
<td>010</td>
</tr>
<tr>
<td>$n_6$</td>
<td>001</td>
<td>000</td>
</tr>
</tbody>
</table>
Solution of the Tutorial Problem 3 for Partial Availability Analysis

Bit vector $a \times b \quad b + c \quad a + b$

<table>
<thead>
<tr>
<th>Node</th>
<th>Local Information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$Gen_n$</td>
<td>$Kill_n$</td>
</tr>
<tr>
<td>$n_1$</td>
<td>110</td>
<td>010</td>
</tr>
<tr>
<td>$n_2$</td>
<td>001</td>
<td>000</td>
</tr>
<tr>
<td>$n_3$</td>
<td>010</td>
<td>000</td>
</tr>
<tr>
<td>$n_4$</td>
<td>001</td>
<td>101</td>
</tr>
<tr>
<td>$n_5$</td>
<td>000</td>
<td>010</td>
</tr>
<tr>
<td>$n_6$</td>
<td>001</td>
<td>000</td>
</tr>
</tbody>
</table>
Solution of the Tutorial Problem 3 for Partial Availability Analysis

Bit vector \( a \times b \mid b + c \mid a + b \)

<table>
<thead>
<tr>
<th>Node</th>
<th>Local Information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( Gen_n ) \mid ( Kill_n ) \mid ( AntGen_n )</td>
<td>Iteration # 1 \mid Changes in iteration # 2</td>
</tr>
<tr>
<td>( n_1 )</td>
<td>110 \mid 010 \mid 100</td>
<td>000 \mid 110</td>
</tr>
<tr>
<td>( n_2 )</td>
<td>001 \mid 000 \mid 001</td>
<td>110 \mid 111</td>
</tr>
<tr>
<td>( n_3 )</td>
<td>010 \mid 000 \mid 010</td>
<td>111 \mid 111</td>
</tr>
<tr>
<td>( n_4 )</td>
<td>001 \mid 101 \mid 000</td>
<td>111 \mid 011</td>
</tr>
<tr>
<td>( n_5 )</td>
<td>000 \mid 010 \mid 000</td>
<td>111 \mid 101</td>
</tr>
<tr>
<td>( n_6 )</td>
<td>001 \mid 000 \mid 001</td>
<td>101 \mid 101</td>
</tr>
</tbody>
</table>
Part 5

Reaching Definitions Analysis
• A definition \( d_x : x = e \) reaches a program point \( p \) if it appears (without a redefinition of \( x \)) on some path from program entry to \( p \) (\( x \) is a variable and \( e \) is an expression)

• Application : Copy Propagation
A use of a variable \( x \) at a program point \( p \) can be replaced by \( y \) if \( d_x : x = y \) is the only definition which reaches \( p \) and \( y \) is not modified between the point of \( d_x \) and \( p \).
Using Reaching Definitions for Def-Use and Use-Def Chains

Def-Use Chains

1. \( a_1: a = 4 \)
   \( b_1: b = 2 \)
   \( c_1: c = 3 \)
   \( n_1: n = c \times 2 \)

2. if \( a > n \)

3. \( a_2: a = a + 1 \)

4. if \( a \geq 12 \)

5. \( t_{l1}: t_1 = a + b \)
   \( a_3: a = t_1 + c \)

6. print \( a \)
Using Reaching Definitions for Def-Use and Use-Def Chains

**Def-Use Chains**

```
a_1: a = 4
b_1: b = 2
c_1: c = 3
n_1: n = c*2
```

```
if (a > n)
```

```
a_2: a = a+1
```

```
if (a ≥ 12)
```

```
t_{1_1}: t_1 = a+b
a_3: a = t_1+c
```

```
print a
```
Using Reaching Definitions for Def-Use and Use-Def Chains

Def-Use Chains

1. \( a_1 : a = 4 \)
2. \( b_1 : b = 2 \)
3. \( c_1 : c = 3 \)
4. \( n_1 : n = c \cdot 2 \)
5. \( \text{if } (a > n) \)
6. \( a_2 : a = a + 1 \)
7. \( \text{if } (a \geq 12) \)
8. \( t_{1_1} : t_1 = a + b \)
9. \( a_3 : a = t_1 + c \)
10. \( \text{print } a \)
Using Reaching Definitions for Def-Use and Use-Def Chains

**Def-Use Chains**

1. \( a_1: a = 4 \)
2. \( b_1: b = 2 \)
3. \( c_1: c = 3 \)
4. \( n_1: n = c \times 2 \)
5. if (a > n)
   - T: \( t_{11}: t_1 = a + b \)
   - F: \( a_2: a = a + 1 \)
   - T: \( a_3: a = t_1 + c \)
6. print a

**Use-Def Chains**

1. \( a_1: a = 4 \)
2. \( b_1: b = 2 \)
3. \( c_1: c = 3 \)
4. \( n_1: n = c \times 2 \)
5. if (a > n)
   - T: \( a_2: a = a + 1 \)
   - F: \( t_{11}: t_1 = a + b \)
   - T: \( a_3: a = t_1 + c \)
6. print a
Using Reaching Definitions for Def-Use and Use-Def Chains

**Def-Use Chains**

1. \( a_1: a = 4 \)
2. \( b_1: b = 2 \)
3. \( c_1: c = 3 \)
4. \( n_1: n = c \times 2 \)
5. if \( (a > n) \)
6. \( a_2: a = a + 1 \)
7. if \( (a \geq 12) \)
8. \( t_1: t_1 = a + b \)
9. \( a_3: a = t_1 + c \)
10. print \( a \)

**Use-Def Chains**

1. \( a_1: a = 4 \)
2. \( b_1: b = 2 \)
3. \( c_1: c = 3 \)
4. \( n_1: n = c \times 2 \)
5. if \( (a > n) \)
6. \( a_2: a = a + 1 \)
7. if \( (a \geq 12) \)
8. \( t_1: t_1 = a + b \)
9. \( a_3: a = t_1 + c \)
10. print \( a \)
Using Reaching Definitions for Def-Use and Use-Def Chains

**Def-Use Chains**

\[
\begin{align*}
a_1 & : a = 4 \\
b_1 & : b = 2 \\
c_1 & : c = 3 \\
n_1 & : n = c \times 2
\end{align*}
\]

1.

2. if \((a > n)\)

3. \(a_2 : a = a + 1\)

4. if \((a \geq 12)\)

5. \(t_{11} : t_1 = a + b \\
a_3 : a = t_1 + c\)

6. print \(a\)

**Use-Def Chains**

\[
\begin{align*}
a_1 & : a = 4 \\
b_1 & : b = 2 \\
c_1 & : c = 3 \\
n_1 & : n = c \times 2
\end{align*}
\]

1.

2. if \((a > n)\)

3. \(a_2 : a = a + 1\)

4. if \((a \geq 12)\)

5. \(t_{11} : t_1 = a + b \\
a_3 : a = t_1 + c\)

6. print \(a\)

There is a need to distinguish between different occurrences of lexically identical definitions.

Hence a definition is identified by the label of the statement.
Defining Data Flow Analysis for Reaching Definitions Analysis

Let $d_v$ be a definition of variable $v$

$$Gen_n = \{ d_v \mid \text{variable } v \text{ is defined in basic block } n \text{ and this definition is not followed (within } n \text{) by a definition of } v \}$$

$$Kill_n = \{ d_v \mid \text{basic block } n \text{ contains a definition of } v \}$$

<table>
<thead>
<tr>
<th>Entity</th>
<th>Manipulation</th>
<th>Exposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Gen_n$</td>
<td>Definition</td>
<td>Occurrence</td>
</tr>
<tr>
<td>$Kill_n$</td>
<td>Definition</td>
<td>Occurrence</td>
</tr>
</tbody>
</table>
Data Flow Equations for Reaching Definitions Analysis

\[ In_n = \begin{cases} \emptyset & \text{if } n \text{ is Start block} \\ \bigcup_{p \in \text{pred}(n)} Out_p & \text{otherwise} \end{cases} \]

\[ Out_n = Gen_n \cup (In_n - Kill_n) \]

\[ BI = \{ d_x : x = \text{undef} \mid x \in \text{Var} \} \]

**In*_n and Out*_n are sets of definitions**
Tutorial Problem for Copy Propagation

1: 

\[ a_1: a = 4 \]
\[ b_1: b = 2 \]
\[ c_1: c = 3 \]
\[ n_1: n = c \times 2 \]

2: if (a > n)

3: 

\[ a_2: a = a + 1 \]

4: if (a ≥ 12)

5: 

\[ t_{11}: t_1 = a + b \]
\[ a_3: a = t_1 + c \]

6: print a
Tutorial Problem for Copy Propagation

1. \( a_1: a = 4 \)
2. \( b_1: b = 2 \)
3. \( c_1: c = 3 \)
4. \( n_1: n = c \times 2 \)

if \( a > n \)

1. \( a = a + 1 \)
2. \( b = 2 \)
3. \( c = 3 \)
4. \( n = c \times 2 \)
5. \( t_{1_1}: t_1 = a + b \)
6. \( t_{1_2}: t_1 = a + b \)

\( a_2: a = a + 1 \)

Local copy propagation and constant folding

if \( a \geq 12 \)

1. \( a = a + 1 \)
2. \( b = 2 \)
3. \( c = 3 \)
4. \( n = c \times 2 \)
5. \( t_{1_1}: t_1 = a + b \)
6. \( t_{1_2}: t_1 = a + b \)

3. \( a_2: a = a + 1 \)
4. \( b = 2 \)
5. \( c = 3 \)
6. \( n = c \times 2 \)

3. \( t_{1_1}: t_1 = a + b \)
4. \( a_3: a = t_1 + c \)
5. \( t_{1_2}: t_1 = a + b \)
6. \( print \ a \)
Tutorial Problem for Copy Propagation

1. \(a_1: a = 4\)
   \(b_1: b = 2\)
   \(c_1: c = 3\)
   \(n_1: n = c \times 2\)

2. \(\text{if } (a > n)\)
   \(T\) → \(F\)

3. \(a_2: a = a + 1\)

4. \(\text{if } (a \geq 12)\)
   \(T\) → \(F\)

5. \(t_1: t_1 = a + b\)
   \(a_3: a = t_1 + c\)

6. \(\text{print } a\)

Local copy propagation and constant folding

1. \(a_1: a = 4\)
   \(b_1: b = 2\)
   \(c_1: c = 3\)
   \(n_1: n = 6\)

2. \(\text{if } (a > n)\)
   \(T\) → \(F\)

3. \(a_2: a = a + 1\)

4. \(\text{if } (a \geq 12)\)
   \(T\) → \(F\)

5. \(t_1: t_1 = a + b\)
   \(a_3: a = t_1 + c\)

6. \(\text{print } a\)
Tutorial Problem for Copy Propagation

1. \( a_1: a = 4 \)
   \( b_1: b = 2 \)
   \( c_1: c = 3 \)
   \( n_1: n = 6 \)

2. \( \text{if } (a > n) \)

3. \( a_2: a = a + 1 \)

4. \( \text{if } (a \geq 12) \)

5. \( t_{1_1}: t_1 = a + b \)
   \( a_3: a = t_1 + c \)

6. \( \text{print } a \)

**Gen** | **Kill**
---|---
\( n_1 \) | \( \{a_1, b_1, c_1, n_1\} \)
\( \{a_0, a_1, a_2, a_3, b_0, b_1, c_0, c_1, n_0, n_1\} \)
\( n_2 \) | \( \emptyset \)
\( \emptyset \)
\( n_3 \) | \( \{a_2\} \)
\( \{a_0, a_1, a_2, a_3\} \)
\( n_4 \) | \( \emptyset \)
\( \emptyset \)
\( n_5 \) | \( \{a_3\} \)
\( \{a_0, a_1, a_2, a_3\} \)
\( n_6 \) | \( \emptyset \)
\( \emptyset \)
Tutorial Problem for Copy Propagation

1. a₁: a = 4  
   b₁: b = 2  
   c₁: c = 3  
   n₁: n = 6

2. if (a > n)
   F
   3. a₂: a = a+1
   T

3. if (a ≥ 12)
   F
   5. t₁₁: t₁ = a+b  
      a₃: a = t₁+c
   T

4. if (a ≥ 12)
   T

5. print a

<table>
<thead>
<tr>
<th></th>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>n₁</td>
<td>{a₁, b₁, c₁, n₁}</td>
<td>{a₀, a₁, a₂, a₃, b₀, b₁, c₀, c₁, n₀, n₁}</td>
</tr>
<tr>
<td>n₂</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>n₃</td>
<td>{a₂}</td>
<td>{a₀, a₁, a₂, a₃}</td>
</tr>
<tr>
<td>n₄</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>n₅</td>
<td>{a₃}</td>
<td>{a₀, a₁, a₂, a₃}</td>
</tr>
<tr>
<td>n₆</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>

- Temporary variable t₁ is ignored
- For variable v, v₀ denotes the definition v = ?
  This is used for defining BI
Tutorial Problem for Copy Propagation

1. \( a_1: a = 4 \)
   \( b_1: b = 2 \)
   \( c_1: c = 3 \)
   \( n_1: n = 6 \)

2. if (a \(>\) n)

3. \( a_2: a = a + 1 \)

4. if (a \(\geq\) 12)

5. \( t_{l1}: t_{l1} = a + b \)
   \( a_3: a = t_{l1} + c \)

6. print a

---

<table>
<thead>
<tr>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>({a_1, b_1, c_1, n_1})</td>
<td>({a_0, a_1, a_2, a_3, b_0, b_1, c_0, c_1, n_0, n_1})</td>
</tr>
<tr>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>({a_2})</td>
<td>({a_0, a_1, a_2, a_3})</td>
</tr>
<tr>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>({a_3})</td>
<td>({a_0, a_1, a_2, a_3})</td>
</tr>
<tr>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Iteration #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{In})</td>
</tr>
<tr>
<td>(n_1)</td>
</tr>
<tr>
<td>(n_2)</td>
</tr>
<tr>
<td>(n_3)</td>
</tr>
<tr>
<td>(n_4)</td>
</tr>
<tr>
<td>(n_5)</td>
</tr>
<tr>
<td>(n_6)</td>
</tr>
</tbody>
</table>
**Tutorial Problem for Copy Propagation**

### Variables
- $a_1$: $a = 4$
- $b_1$: $b = 2$
- $c_1$: $c = 3$
- $n_1$: $n = 6$

### Program

1. \[a_1: a = 4\]
2. \[b_1: b = 2\]
3. \[c_1: c = 3\]
4. \[n_1: n = 6\]

#### Iteration #2

<table>
<thead>
<tr>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_1$</td>
<td>${a_0, a_1, a_2, a_3, b_0, b_1, c_0, c_1, n_0, n_1}$</td>
</tr>
<tr>
<td>$n_2$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$n_3$</td>
<td>${a_0, a_1, a_2, a_3}$</td>
</tr>
<tr>
<td>$n_4$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$n_5$</td>
<td>${a_0, a_1, a_2, a_3}$</td>
</tr>
<tr>
<td>$n_6$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

#### Code

2. `if (a > n)`

\[
F \quad a_2: a = a + 1
\]

4. `if (a ≥ 12)`

\[
T \quad t_1: t_1 = a + b
\]

\[
a_3: a = t_1 + c
\]

6. `print a`
Tutorial Problem for Copy Propagation

1

\begin{align*}
a_1 &: a = 4 \\
b_1 &: b = 2 \\
c_1 &: c = 3 \\
n_1 &: n = 6
\end{align*}

2

\textbf{if} (a > n)

3

\begin{align*}
a_2 &: a = a + 1
\end{align*}

\begin{align*}
\{a_1, a_2, b_1, c_1, n_1\}
\end{align*}

4

\textbf{if} (a \geq 12)

5

\textbf{if} (\text{true})

6

\textbf{print} a

\begin{align*}
t_{11} &: t1 = a + b \\
a_3 &: a = t1 + c
\end{align*}

\begin{align*}
\{a_1, a_2, a_3, b_1, c_1, n_1\}
\end{align*}
Tutorial Problem for Copy Propagation

1

\[
\begin{align*}
a_1 & : a = 4 \\
b_1 & : b = 2 \\
c_1 & : c = 3 \\
n_1 & : n = 6
\end{align*}
\]

\[\{a_1, a_2, b_1, c_1, n_1\}\]

2

\[\text{if } (a > n)\]

\[F \{a_1, a_2, b_1, c_1, n_1\}\]

3

\[a_2 : a = a + 1\]

\[\{a_1, a_2, b_1, c_1, n_1\}\]

4

\[\text{if } (a \geq 12)\]

\[F \{a_1, a_2, b_1, c_1, n_1\}\]

5

\[t_{l_1} : t_1 = a + b\]

\[a_3 : a = t_1 + c\]

\[\{a_1, a_2, a_3, b_1, c_1, n_1\}\]

6

\[\text{print } a\]

- RHS of \(n_1\) is constant and hence cannot change
### Tutorial Problem for Copy Propagation

1. \(a_1: a = 4\)  
   \(b_1: b = 2\)  
   \(c_1: c = 3\)  
   \(n_1: n = 6\)

2. if \((a > 6)\)  
   \(F\) \(\{a_1, a_2, b_1, c_1, n_1\}\)  

3. \(a_2: a = a + 1\)
   \(\{a_1, a_2, b_1, c_1, n_1\}\)

4. if \((a \geq 12)\)  
   \(T\) \(F\) \(\{a_1, a_2, b_1, c_1, n_1\}\)

5. \(t_1: t_1 = a + b\)  
   \(a_3: a = t_1 + c\)
   \(\{a_1, a_2, a_3, b_1, c_1, n_1\}\)

6. print \(a\)

- RHS of \(n_1\) is constant and hence cannot change
- In block 2, \(n\) can be replaced by 6
Tutorial Problem for Copy Propagation

1

\begin{align*}
a_1 & : a = 4 \\
b_1 & : b = 2 \\
c_1 & : c = 3 \\
n_1 & : n = 6 \\
\end{align*}

\{a_1, a_2, b_1, c_1, n_1\}

2

if (a > 6)

\begin{align*}
a_2 & : a = a + 1 \\
\end{align*}

\{a_1, a_2, b_1, c_1, n_1\}

3

4

if (a >= 12)

\begin{align*}
t_{1_1} & : t_1 = a + b \\
a_3 & : a = t_1 + c \\
\end{align*}

\{a_1, a_2, a_3, b_1, c_1, n_1\}

5

6

print a

\{a_1, a_2, a_3, b_1, c_1, n_1\}

- RHS of \( n_1 \) is constant and hence cannot change
- In block 2, \( n \) can be replaced by 6
- RHS of \( b_1 \) and \( c_1 \) are constant and hence cannot change
Tutorial Problem for Copy Propagation

1. \( a_1: a = 4 \)
   \( b_1: b = 2 \)
   \( c_1: c = 3 \)
   \( n_1: n = 6 \)

2. if \( a > 6 \)
   \( F \{a_1, a_2, b_1, c_1, n_1\} \)

3. \( a_2: a = a + 1 \)
   \( \{a_1, a_2, b_1, c_1, n_1\} \)

4. if \( a \geq 12 \)
   \( F \{a_1, a_2, b_1, c_1, n_1\} \)

5. \( t_{l1}: t1 = a + 2 \)
   \( a_3: a = t1 + 3 \)
   \( \{a_1, a_2, a_3, b_1, c_1, n_1\} \)

6. print \( a \)

- RHS of \( n_1 \) is constant and hence cannot change
- In block 2, \( n \) can be replaced by 6
- RHS of \( b_1 \) and \( c_1 \) are constant and hence cannot change
- In block 5, \( b \) can be replaced by 2 and \( c \) can be replaced by 3
Tutorial Problem for Copy Propagation

1

\[
a_1: a = 4 \\
b_1: b = 2 \\
c_1: c = 3 \\
n_1: n = 6
\]

2

\[ \text{if } (a > 6) \]

3

\[ a_2: a = a + 1 \]

\{a\}

4

\[ \text{if } (a \geq 12) \]

5

\[ t_{l1}: t1 = a + 2 \\
a_3: a = t1 + 3 \]

6

\text{print } a
Tutorial Problem for Copy Propagation

1

\[ a_1: \ a = 4 \]
\[ b_1: \ b = 2 \]
\[ c_1: \ c = 3 \]
\[ n_1: \ n = 6 \]

2

\[ \text{if } (a > 6) \]

3

\[ a_2: \ a = a + 1 \]

4

\[ \{a\} \]

\[ \text{if } (a \geq 12) \]

5

\[ t_{11}: \ t_1 = a + 2 \]
\[ a_3: \ a = t_1 + 3 \]

6

\[ \text{print } a \]

So what is the advantage?
Tutorial Problem for Copy Propagation

1

\[ a_1: a = 4 \]
\[ b_1: b = 2 \]
\[ c_1: c = 3 \]
\[ n_1: n = 6 \]

2

\( \text{if } (a > 6) \)

3

\[ a_2: a = a + 1 \]

4

\( \text{if } (a \geq 12) \)

5

\[ t_{1_1}: t_1 = a + 2 \]
\[ a_3: a = t_1 + 3 \]

6

print a

So what is the advantage?

Dead Code Elimination
Tutorial Problem for Copy Propagation

\begin{align*}
a_1 & : a = 4 \\
b_1 & : b = 2 \\
c_1 & : c = 3 \\
n_1 & : n = 6
\end{align*}

1. \{a\}

2. if (a > 6)
   \begin{align*}
a_2 & : a = a + 1
\end{align*}

3. \{a\}

4. if (a \geq 12)
   \begin{align*}
t_{11} & : t_1 = a + 2 \\
a_3 & : a = t_1 + 3
\end{align*}

5. \{a\}

6. print a

So what is the advantage?

Dead Code Elimination

- Only \(a\) is live at the exit of 1
Tutorial Problem for Copy Propagation

1. \(a_1: a = 4\)
2. \(b_1: b = 2\)
3. \(c_1: c = 3\)
4. \(n_1: n = 6\)

\{a\}

2. \(\text{if } (a>6)\)
3. \(a_2: a = a + 1\)
   \{a\}

4. \(\text{if } (a \geq 12)\)
5. \(t_1: t_1 = a + 2\)
6. \(a_3: a = t_1 + 3\)

\(\text{print } a\)

So what is the advantage?

Dead Code Elimination

- Only \(a\) is live at the exit of 1
- Assignments of \(b\), \(c\), and \(n\) are dead code
Tutorial Problem for Copy Propagation

So what is the advantage?

Dead Code Elimination

- Only \( a \) is live at the exit of 1
- Assignments of \( b, c, \) and \( n \) are dead code
- Can be deleted
Part 6

Anticipable Expressions Analysis
Defining Anticipable Expressions Analysis

• An expression $e$ is anticipable at a program point $p$, if every path from $p$ to the program exit contains an evaluation of $e$ which is not preceded by a redefinition of any operand of $e$.

• Application: Safety of Code Placement
Safety of Code Placement

Placing $a/b$ at the exit of 1 is unsafe ($\equiv$ can change the behaviour of the optimized program)
Placing $a/b$ at the exit of 1 is unsafe ($\equiv$ can change the behaviour of the optimized program)

Placing $a/b$ at the exit of 1 is unsafe ($\equiv$ can change the behaviour of the optimized program)
Safety of Code Placement

1. if \( b == 0 \)

   False
   2. \( c = a/b \)
   3. False

   True
   1. \( f(a/b) \)
   3. True

Placing \( a/b \) at the exit of 1 is unsafe (\( \equiv \) can change the behaviour of the optimized program)

A guarded computation of an expression should not be converted to an unguarded computation
Defining Data Flow Analysis for Anticipable Expressions Analysis

\[ Gen_n = \{ e \mid \text{expression } e \text{ is evaluated in basic block } n \text{ and} \]
\[ \text{this evaluation is not preceded (within } n) \text{ by a} \]
\[ \text{definition of any operand of } e \} \]

\[ Kill_n = \{ e \mid \text{basic block } n \text{ contains a definition of an operand of } e \} \]

<table>
<thead>
<tr>
<th>Entity</th>
<th>Manipulation</th>
<th>Exposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Gen_n )</td>
<td>Expression</td>
<td>Use</td>
</tr>
<tr>
<td>( Kill_n )</td>
<td>Expression</td>
<td>Modification</td>
</tr>
</tbody>
</table>
Data Flow Equations for Anticipable Expressions Analysis

\[ \text{In}_n = \text{Gen}_n \cup (\text{Out}_n - \text{Kill}_n) \]

\[ \text{Out}_n = \begin{cases} 
\bigcap_{s \in \text{succ}(n)} \text{In}_s & \text{otherwise} \\
\text{BI} & \text{n is End block}
\end{cases} \]

*In\(_n\) and Out\(_n\) are sets of expressions*
Tutorial Problem 1 for Anticipable Expressions Analysis

$$\text{Expr} = \{ \ a \times b, \ b + c, \ b - c \ \}$$
## Solution of Tutorial Problem 1

<table>
<thead>
<tr>
<th>Block</th>
<th>Local Information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$Gen_n$</td>
<td>$Kill_n$</td>
</tr>
<tr>
<td>$n_6$</td>
<td>110</td>
<td>000</td>
</tr>
<tr>
<td>$n_5$</td>
<td>001</td>
<td>000</td>
</tr>
<tr>
<td>$n_4$</td>
<td>010</td>
<td>011</td>
</tr>
<tr>
<td>$n_3$</td>
<td>010</td>
<td>100</td>
</tr>
<tr>
<td>$n_2$</td>
<td>001</td>
<td>011</td>
</tr>
<tr>
<td>$n_1$</td>
<td>000</td>
<td>111</td>
</tr>
</tbody>
</table>
## Solution of Tutorial Problem 1

### Local Information

<table>
<thead>
<tr>
<th>Block</th>
<th>$Gen_n$</th>
<th>$Kill_n$</th>
<th>$Out_n$</th>
<th>$In_n$</th>
<th>$Out_n$</th>
<th>$In_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_6$</td>
<td>110</td>
<td>000</td>
<td>000</td>
<td>110</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n_5$</td>
<td>001</td>
<td>000</td>
<td>110</td>
<td>111</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n_4$</td>
<td>010</td>
<td>011</td>
<td>111</td>
<td>110</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n_3$</td>
<td>010</td>
<td>100</td>
<td>110</td>
<td>010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n_2$</td>
<td>001</td>
<td>011</td>
<td>010</td>
<td>001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n_1$</td>
<td>000</td>
<td>111</td>
<td>001</td>
<td>000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Solution of Tutorial Problem 1

<table>
<thead>
<tr>
<th>Block</th>
<th>Local Information</th>
<th>Global Information</th>
<th>Iteration # 1</th>
<th>Change in iteration # 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$Gen_n$</td>
<td>$Kill_n$</td>
<td>$Out_n$</td>
<td>$In_n$</td>
</tr>
<tr>
<td>$n_6$</td>
<td>110</td>
<td>000</td>
<td>000</td>
<td>110</td>
</tr>
<tr>
<td>$n_5$</td>
<td>001</td>
<td>000</td>
<td>110</td>
<td>111</td>
</tr>
<tr>
<td>$n_4$</td>
<td>010</td>
<td>011</td>
<td>111</td>
<td>110</td>
</tr>
<tr>
<td>$n_3$</td>
<td>010</td>
<td>100</td>
<td>110</td>
<td>010</td>
</tr>
<tr>
<td>$n_2$</td>
<td>001</td>
<td>011</td>
<td>010</td>
<td>001</td>
</tr>
<tr>
<td>$n_1$</td>
<td>000</td>
<td>111</td>
<td>001</td>
<td>000</td>
</tr>
</tbody>
</table>

Jul 2017
Tutorial Problem 2 for Anticipable Expressions Analysis

\[
\begin{align*}
&n_1 \quad d = a \times b; \\
&\quad \text{if}(d) \\
&n_2 \quad a = a \times b; \\
&n_3 \quad c = a \times b; \\
&n_4 \quad \text{if}(c) \\
&n_5 \quad d = c + d; \\
&\quad a = 5; \\
&n_6 \quad \text{print } a \times b; \\
\end{align*}
\]

\[\mathcal{E}xpr = \{a \times b, c + d\}\]
## Solution of Tutorial Problem 2

<table>
<thead>
<tr>
<th>Block</th>
<th>Local Information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$Gen_n$</td>
<td>$Kill_n$</td>
</tr>
<tr>
<td>$n_6$</td>
<td>10</td>
<td>00</td>
</tr>
<tr>
<td>$n_5$</td>
<td>01</td>
<td>11</td>
</tr>
<tr>
<td>$n_4$</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>$n_3$</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>$n_2$</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>$n_1$</td>
<td>10</td>
<td>01</td>
</tr>
</tbody>
</table>
## Solution of Tutorial Problem 2

<table>
<thead>
<tr>
<th>Block</th>
<th>Local Information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( Gen_n )</td>
<td>( Kill_n )</td>
</tr>
<tr>
<td>( n_6 )</td>
<td>10</td>
<td>00</td>
</tr>
<tr>
<td>( n_5 )</td>
<td>01</td>
<td>11</td>
</tr>
<tr>
<td>( n_4 )</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>( n_3 )</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>( n_2 )</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>( n_1 )</td>
<td>10</td>
<td>01</td>
</tr>
</tbody>
</table>
# Solution of Tutorial Problem 2

<table>
<thead>
<tr>
<th>Block</th>
<th>Local Information</th>
<th>Global Information</th>
<th>Iteration # 1</th>
<th>Change in iteration # 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$Gen_n$</td>
<td>$Kill_n$</td>
<td>$Out_n$</td>
<td>$In_n$</td>
</tr>
<tr>
<td>$n_6$</td>
<td>10</td>
<td>00</td>
<td>00</td>
<td>10</td>
</tr>
<tr>
<td>$n_5$</td>
<td>01</td>
<td>11</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>$n_4$</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>$n_3$</td>
<td>10</td>
<td>01</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>$n_2$</td>
<td>10</td>
<td>10</td>
<td>01</td>
<td>11</td>
</tr>
<tr>
<td>$n_1$</td>
<td>10</td>
<td>01</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>
Part 7

Common Features of Bit Vector Data Flow Frameworks
Defining Local Data Flow Properties

- Live variables analysis

<table>
<thead>
<tr>
<th>Entity</th>
<th>Manipulation</th>
<th>Exposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Gen_n$</td>
<td>Variable</td>
<td>Use</td>
</tr>
<tr>
<td>$Kill_n$</td>
<td>Variable</td>
<td>Modification</td>
</tr>
</tbody>
</table>

- Analysis of expressions

<table>
<thead>
<tr>
<th>Entity</th>
<th>Manipulation</th>
<th>Exposition</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Exposition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Availability</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Anticipability</td>
</tr>
<tr>
<td>$Gen_n$</td>
<td>Expression</td>
<td>Use</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Kill_n$</td>
<td>Expression</td>
<td>Modification</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Common Form of Data Flow Equations

\[ X_i = f(Y_i) \]
\[ Y_i = \bigcap X_j \]
Common Form of Data Flow Equations

So far we have seen sets (or bit vectors). Could be entities other than sets.

\[ X_i = f(Y_i) \]
\[ Y_i = \bigcap X_j \]
Common Form of Data Flow Equations

Data Flow Information

So far we have seen sets (or bit vectors). Could be entities other than sets.

Flow Function

So far we have seen constant Gen and Kill. Could be dependent Gen and Kill.

\[ X_i = f(Y_i) \]

\[ Y_i = \bigcap X_j \]
Common Form of Data Flow Equations

So far we have seen sets (or bit vectors). Could be entities other than sets.

\[
\begin{align*}
X_i &= f(Y_i) \\
Y_i &= \cap X_j
\end{align*}
\]

So far we have seen constant $Gen$ and $Kill$. Could be dependent $Gen$ and $Kill$.

So far we have seen $\cup$ and $\cap$. Could be other operations.
## A Taxonomy of Bit Vector Data Flow Frameworks

<table>
<thead>
<tr>
<th></th>
<th>Confluence</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Union</td>
</tr>
<tr>
<td>Forward</td>
<td>Reaching Definitions</td>
</tr>
<tr>
<td>Backward</td>
<td>Live Variables</td>
</tr>
<tr>
<td>Bidirectional (limited)</td>
<td>Partial Redundancy Elimination</td>
</tr>
<tr>
<td></td>
<td>(Original M-R Formulation)</td>
</tr>
<tr>
<td></td>
<td>Intersection</td>
</tr>
<tr>
<td></td>
<td>Available Expressions</td>
</tr>
<tr>
<td></td>
<td>Anticipable Expressions</td>
</tr>
</tbody>
</table>

Jul 2017 IIT Bombay
# A Taxonomy of Bit Vector Data Flow Frameworks

<table>
<thead>
<tr>
<th></th>
<th>Confluence</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Union</td>
</tr>
<tr>
<td>Forward</td>
<td>Reaching Definitions</td>
</tr>
<tr>
<td>Backward</td>
<td>Live Variables</td>
</tr>
<tr>
<td>Bidirectional</td>
<td>Partial Redundancy Elimination (Original M-R Formulation)</td>
</tr>
<tr>
<td>(limited)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Intersection</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Available Expressions</td>
</tr>
<tr>
<td></td>
<td>Anticipable Expressions</td>
</tr>
</tbody>
</table>

Any Path
## A Taxonomy of Bit Vector Data Flow Frameworks

<table>
<thead>
<tr>
<th></th>
<th>Union</th>
<th>Intersection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward</td>
<td>Reaching Definitions</td>
<td>Available Expressions</td>
</tr>
<tr>
<td>Backward</td>
<td>Live Variables</td>
<td>Anticipable Expressions</td>
</tr>
<tr>
<td>Bidirectional (limited)</td>
<td></td>
<td>Partial Redundancy Elimination (Original M-R Formulation)</td>
</tr>
</tbody>
</table>

- **Any Path**
- **All Paths**
### A Taxonomy of Bit Vector Data Flow Frameworks

<table>
<thead>
<tr>
<th></th>
<th>Confluence</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Union</td>
</tr>
<tr>
<td>Forward</td>
<td>Reaching Definitions</td>
</tr>
<tr>
<td>Backward</td>
<td>Live Variables</td>
</tr>
<tr>
<td>Bidirectional (limited)</td>
<td></td>
</tr>
</tbody>
</table>
A Taxonomy of Bit Vector Data Flow Frameworks

<table>
<thead>
<tr>
<th></th>
<th>Confluence</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Union</td>
</tr>
<tr>
<td>Forward</td>
<td>Reaching Definitions</td>
</tr>
<tr>
<td>Backward</td>
<td>Live Variables</td>
</tr>
<tr>
<td>Bidirectional (limited)</td>
<td></td>
</tr>
</tbody>
</table>
# A Taxonomy of Bit Vector Data Flow Frameworks

<table>
<thead>
<tr>
<th></th>
<th>Confluence</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Union</td>
</tr>
<tr>
<td>Forward</td>
<td>Reaching Definitions</td>
</tr>
<tr>
<td>Backward</td>
<td>Live Variables</td>
</tr>
<tr>
<td>Bidirectional</td>
<td></td>
</tr>
<tr>
<td>(limited)</td>
<td></td>
</tr>
</tbody>
</table>

**Any Path**

**All Paths**
Data Flow Paths Discovered by Data Flow Analysis

Liveness

Anticipability

Availability

Partial Availability

\[ a \times b \]

\[ a \times b \]

\[ a \times b \]

\[ a \times b \]
Data Flow Paths Discovered by Data Flow Analysis

Sequence of blocks \((n_1, n_2, \ldots, n_k)\) which is a prefix of some potential execution path starting at \(n_1\) such that:

- \(n_k\) contains an upwards exposed use of \(v\), and
- no other block on the path contains an assignment to \(v\).

Liveness
Data Flow Paths Discovered by Data Flow Analysis

Sequence of blocks \((n_1, n_2, \ldots, n_k)\) which is a prefix of some potential execution path starting at \(n_1\) such that:

- \(n_k\) contains an upwards exposed use of \(a \times b\), and
- no other block on the path contains an assignment to \(a\) or \(b\), and
- every path starting at \(n_1\) is an anticipability path of \(a \times b\).
Data Flow Paths Discovered by Data Flow Analysis

Sequence of blocks \((n_1, n_2, \ldots, n_k)\) which is a prefix of some potential execution path starting at \(n_1\) such that:

- \(n_1\) contains a downwards exposed use of \(a \times b\), and
- no other block on the path contains an assignment to \(a\) or \(b\), and
- every path ending at \(n_k\) is an availability path of \(a \times b\).
Data Flow Paths Discovered by Data Flow Analysis

Sequence of blocks \((n_1, n_2, \ldots, n_k)\) which is a prefix of some potential execution path starting at \(n_1\) such that:

- \(n_1\) contains a downwards exposed use of \(a \ast b\), and
- no other block on the path contains an assignment to \(a\) or \(b\).
Data Flow Paths Discovered by Data Flow Analysis

Liveness

Anticipability

Availability

Partial Availability
Partial Redundancy Elimination
**Precursor: Common Subexpression Elimination**

<table>
<thead>
<tr>
<th>Code Fragment</th>
<th>Flow Graph</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>if (...)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>c = a*b;</td>
<td></td>
</tr>
<tr>
<td>else</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>d = a*b;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>e = a*b;</td>
<td></td>
</tr>
</tbody>
</table>
## Precursor: Common Subexpression Elimination

<table>
<thead>
<tr>
<th>Code Fragment</th>
<th>Flow Graph</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>if (...)</code></td>
<td><img src="image" alt="Flow Graph" /></td>
<td><code>a</code> and <code>b</code> are not modified along paths 1 → 2 → 4 and 1 → 3 → 4</td>
</tr>
<tr>
<td><code>c = a*b;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>else</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>d = a*b;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>e = a*b;</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Precursor: Common Subexpression Elimination

<table>
<thead>
<tr>
<th>Code Fragment</th>
<th>Flow Graph</th>
<th>Remarks</th>
</tr>
</thead>
</table>
| if (...)      | ![Flow Graph](image) | • $a$ and $b$ are not modified along paths $1 \rightarrow 2 \rightarrow 4$ and $1 \rightarrow 3 \rightarrow 4$  
• Computation of $a \times b$ in 4 is redundant |
| $c = a*b;$    |            |         |
| else          |            |         |
| $d = a*b;$    |            |         |
| $e = a*b;$    |            |         |
Precursor: Common Subexpression Elimination

<table>
<thead>
<tr>
<th>Code Fragment</th>
<th>Flow Graph</th>
<th>Remarks</th>
</tr>
</thead>
</table>
| if (...)      | ![Flow Graph](image) | • $a$ and $b$ are not modified along paths 1 → 2 → 4 and 1 → 3 → 4  
• Computation of $a \times b$ in 4 is redundant  
• Previous value can be used |
| $c = a \times b$; | 2  | 3  | $d = a \times b$ |
| else          | 4  |  | $e = a \times b$ |
| $d = a \times b$; |  |  |  |
| $e = a \times b$; |  |  |  |
## Precursor: Common Subexpression Elimination

<table>
<thead>
<tr>
<th>Code Fragment</th>
<th>Flow Graph</th>
<th>Remarks</th>
</tr>
</thead>
</table>
| if (...)      | ![Flow Graph Diagram](attachment:flow_graph.png) | • $a$ and $b$ are not modified along paths $1 \rightarrow 2 \rightarrow 4$ and $1 \rightarrow 3 \rightarrow 4$  
• Computation of $a \times b$ in 4 is redundant  
• Previous value can be used |
| $c = a \times b$; |                         |         |
| else          |                         |         |
| $d = a \times b$; |                         |         |
| $e = a \times b$; |                         |         |
Partial Redundancy Elimination

1. if (...)
2. \( a * b \)
3. \( a = 5 \)
4. \( a * b \)
Partial Redundancy Elimination

1. if (...)  
2. $a \times b$  
3. $a = 5$  
4. $a \times b$

- Computation of $a \times b$ in 4 is
Partial Redundancy Elimination

Computation of $a \ast b$ in 4 is redundant along path $1 \rightarrow 2 \rightarrow 4$, but ...
Partial Redundancy Elimination

- Computation of $a \times b$ in 4 is
  - redundant along path $1 \rightarrow 2 \rightarrow 4$, but ...  
  - not redundant along path $1 \rightarrow 3 \rightarrow 4$
Code Hoisting for Partial Redundancy Elimination

1. if (...)  
2. $a \times b$  
3. $a = 5$  
4. $a \times b$
Code Hoisting for Partial Redundancy Elimination

- Computation of $a \times b$ in 3 becomes totally redundant
- Can be deleted
PRE Subsumes Loop Invariant Movement
PRE Subsumes Loop Invariant Movement

What's that?

1

\[ a = b \times c \]

2

3
What's that?

Translate to
PRE Subsumes Loop Invariant Movement

What's that?

1

2

a = b \times c

Translate to

1
t = b \times c

2

a = t

3
PRE Subsumes Loop Invariant Movement

1

2

\[ a = b \times c \]

3
PRE Subsumes Loop Invariant Movement

\[
a = b \times c
\]

1

2

3

1

2

\[
a = b \times c
\]
PRE Subsumes Loop Invariant Movement

\[
a = b \times c
\]

1

2

3

\[
t = b \times c
\]

1

2

3

\[
a = t
\]
PRE Can be Used for Strength Reduction

\[ i = 0 \]

\[ t_1 = i \times 4 \]
\[ a = A[t_1] \]
\[ i = i + 1 \]
PRE Can be Used for Strength Reduction

- * in the loop has been replaced by +
- $i = i + 1$ in the loop has been eliminated
PRE Can be Used for Strength Reduction

- Delete \( i = i + 1 \)

\[
i = 0
\]

\[
t_1 = i \times 4
\]

\[
a = A[t_1]
\]

\[
i = i + 1
\]
PRE Can be Used for Strength Reduction

- $i = 0$

- $t_1 = i \times 4$
  $a = A[t_1]$
  $i = i + 1$

- Delete $i = i + 1$

- Expression $i \times 4$ becomes loop invariant
PRE Can be Used for Strength Reduction

\[ i = 0 \]
\[ t_1 = i \times 4 \]

- Delete \( i = i + 1 \)
- Expression \( i \times 4 \) becomes loop invariant
- Hoist it and increment \( t_1 \) in the loop

\[ a = A[t_1] \]
\[ t_1 = t_1 + 4 \]
PRE Can be Used for Strength Reduction

\[
\begin{align*}
  i &= 0 \\
  t1 &= i \times 4
\end{align*}
\]

\[
\begin{align*}
  a &= A[t1] \\
  t1 &= t1 + 4
\end{align*}
\]

- Delete \( i = i + 1 \)
- Expression \( i \times 4 \) becomes loop invariant
- Hoist it and increment \( t1 \) in the loop

- \( \times \) in the loop has been replaced by +
- \( i = i + 1 \) in the loop has been eliminated
Performing Partial Redundancy Elimination

1. Identify partial redundancies
2. Identify program points where computations can be inserted
3. Insert expressions
4. Partial redundancies become total redundancies
   \[ \implies \text{Delete them.} \]

Morel-Renvoise Algorithm \((CACM, 1979.)\)
Defining Hoisting Criteria

• An expression can be safely inserted at a program point $p$ if it is

![Diagram showing available expressions at $p$]
Defining Hoisting Criteria

- An expression can be safely inserted at a program point $p$ if it is
  
  **Available at $p$**

  ![Diagram showing expression available at $p$]

  **Anticipable at $p$**

  ![Diagram showing expression anticipable at $p$]
Defining Hoisting Criteria

- An expression can be safely inserted at a program point $p$ if it is
  
  **Available at $p$**
  
  ![Diagram showing expression available at $p$]
  
  **Anticipable at $p$**
  
  ![Diagram showing expression anticipable at $p$]
  
  - If it is available at $p$, then there is no need to insert it at $p$. 

Defining Hoisting Criteria

- An expression can be safely inserted at a program point $p$ if it is

- **Available at $p$**
- **Anticipable at $p$**

- If it is available at $p$, then there is no need to insert it at $p$.
- If it is anticipable at $p$ then all such occurrences should be hoisted to $p$. 
Defining Hoisting Criteria

- An expression can be safely inserted at a program point $p$ if it is

  Available at $p$
  
  - If it is available at $p$, then there is no need to insert it at $p$.

  Anticipable at $p$
  
  - If it is anticipable at $p$ then all such occurrences should be hoisted to $p$.
  
  - An expression should be hoisted to $p$ provided it can be hoisted to $p$ along all paths from $p$ to exit.
Safety of Hoisting an Expression

Predecessor Blocks

Basic Block

Entry

Exit

Successor Blocks
Safety of Hoisting an Expression

- Safety of hoisting to the exit of a block
Safety of Hoisting an Expression

- *Safety of hoisting to the exit of a block*

- *Safety of hoisting to the entry of a block*
Safety of Hoisting an Expression

- Safety of hoisting to the exit of a block
- Safety of hoisting to the entry of a block
- Safety of hoisting out of the entry of a block
Safety of Hoisting an Expression

- Safety of hoisting to the exit of a block
  
  S.1 Hoist only if it can be hoisted out of the entries of all successor blocks

- Safety of hoisting to the entry of a block

- Safety of hoisting out of the entry of a block
Safety of Hoisting an Expression

- Safety of hoisting to the exit of a block

- Safety of hoisting to the entry of a block
  
  S.2 Hoist only if
  
  S.2.a it is upwards exposed, or

- Safety of hoisting out of the entry of a block
Safety of Hoisting an Expression

- Safety of hoisting to the exit of a block

- Safety of hoisting to the entry of a block
  
  S.2 Hoist only if
  
  S.2.a it is upwards exposed, or
  
  S.2.b it can be hoisted to its exit and is transparent in the block

- Safety of hoisting out of the entry of a block
Safety of Hoisting an Expression

- Safety of hoisting to the exit of a block

- Safety of hoisting to the entry of a block

- Safety of hoisting out of the entry of a block

S.3 Hoist only if for each predecessor
S.3.a it can be hoisted to its exit, or
Safety of Hoisting an Expression

- Safety of hoisting to the exit of a block
- Safety of hoisting to the entry of a block
- Safety of hoisting out of the entry of a block

S.3 Hoist only if for each predecessor
- S.3.a it can be hoisted to its exit, or
- S.3.b it is available at its exit.
Safety of Hoisting an Expression

- **Safety of hoisting to the exit of a block**
  
  S.1 Hoist only if it can be hoisted out of the entries of all successor blocks

- **Safety of hoisting to the entry of a block**
  
  S.2 Hoist only if
  
  S.2.a it is upwards exposed, or
  S.2.b it can be hoisted to its exit and is transparent in the block

- **Safety of hoisting out of the entry of a block**
  
  S.3 Hoist only if for each predecessor
  
  S.3.a it can be hoisted to its exit, or
  S.3.b it is available at its exit.
Safety of Hoisting an Expression

- **Safety of hoisting to the exit of a block**

  S.1 Hoist only if it can be hoisted out of the entries of all successor blocks

- **Safety of hoisting to the entry of a block**

  S.2 Hoist only if
  - S.2.a it is upwards exposed, or
  - S.2.b it can be hoisted to its exit and is transparent in the block

- **Safety of hoisting out of the entry of a block**

  S.3 Hoist only if for each predecessor
  - S.3.a it can be hoisted to its exit, or
  - S.3.b it is available at its exit.
Safety of Hoisting an Expression

- **Safety of hoisting to the exit of a block**
  
  S.1 Hoist only if it can be hoisted out of the entries of all successor blocks.

- **Safety of hoisting to the entry of a block**
  
  S.2 Hoist only if:
  
  S.2.a it is upwards exposed, or
  S.2.b it can be hoisted to its exit and is transparent in the block.

- **Safety of hoisting out of the entry of a block**
  
  S.3 Hoist only if for each predecessor:
  
  S.3.a it can be hoisted to its exit, or
  S.3.b it is available at its exit.
Anticipability and Code Hoisting

- What is the meaning of the assertion

  "\( a \ast b \) is anticipable at program point \( p \)"

  - \( a \ast b \) is computed along every path from \( p \) to \( End \) before \( a \) or \( b \) are modified
  - The value computed at \( p \) would be same as the next value computed on any path
  - \( a \ast b \) can be safely inserted at \( p \)
Anticipability and Code Hoisting

- What is the meaning of the assertion
  “a * b is anticipable at program point p”
  - a * b is computed along every path from p to End before a or b are modified
  - The value computed at p would be same as the next value computed on any path
  - a * b can be safely inserted at p

- It does not say that the subsequent computations of a * b can be deleted
  (Expression may not be available at the subsequent points)
Anticipability and Code Hoisting

- What is the meaning of the assertion “\(a \times b\) is anticipable at program point \(p\)”
  - \(a \times b\) is computed along every path from \(p\) to \(End\) before \(a\) or \(b\) are modified
  - The value computed at \(p\) would be same as the next value computed on any path
  - \(a \times b\) can be safely inserted at \(p\)

- It does not say that the subsequent computations of \(a \times b\) can be deleted
  (Expression may not be available at the subsequent points)

- Hoisting involves
  - making the expressions available and
  - deleting their subsequent computations
A Comparison of Anticipability and Hoistability

Anticipability

Hoistability

\[ a = 5 \]

\[ a \times b \]
A Comparison of Anticipability and Hoistability

Anticipability

\[ a = 5 \]

\[ a \times b \]

Hoistability
A Comparison of Anticipability and Hoistability

Anticipability

```
  a = 5
```

Hoistability

```
a * b
```

0 1
A Comparison of Anticipability and Hoistability

Anticipability

Hoistability

Characterises safety of placement but not safety of hoisting
A Comparison of Anticipability and Hoistability

Anticipability

Hoistability

Characterises safety of placement but not safety of hoisting
A Comparison of Anticipability and Hoistability

**Anticipability**

- $a = 5$
- $a \ast b$

**Hoistability**

- $a = 5$
- $a \ast b$

Characterises safety of placement but not safety of hoisting
A Comparison of Anticipability and Hoistability

Anticipability

Hoistability

Characterises safety of placement but not safety of hoisting
A Comparison of Anticipability and Hoistability

Anticipability

Hoistability

Characterises safety of placement but not safety of hoisting
A Comparison of Anticipability and Hoistability

Anticipability

Hoistability

Characterises safety of placement but not safety of hoisting
A Comparison of Anticipability and Hoistability

Anticipability

Characterises safety of placement but not safety of hoisting

Hoistability

Characterises safety of hoisting
A Comparison of Anticipability and Hoistability

**Anticipability**
- Characterises safety of placement but not safety of hoisting

**Hoistability**
- Characterises safety of hoisting

*Hoist an expression to the entry of a block only if it can be hoisted out of the block into all predecessor blocks*
Revised Safety Criteria of Hoisting an Expression

- **Safety of hoisting to the exit of a block**
  
  S.1 Hoist only if it can be hoisted out of the entries of all successor blocks

- **Safety of hoisting to the entry of a block**
  
  S.2 Hoist only if
  
  S.2.a it is upwards exposed, or
  S.2.b it can be hoisted to its exit and is transparent in the block

- **Safety of hoisting out of the entry of a block**
  
  S.3 Hoist only if for each predecessor
  
  S.3.a it can be hoisted to its exit, or
  S.3.b it is available at its exit.
Revised Safety Criteria of Hoisting an Expression

- **Safety of hoisting to the exit of a block**
  
  S.1 Hoist only if it can be hoisted out of the entries of all successor blocks

- **Safety of hoisting to the entry of a block**
  
  S.2 Hoist only if
  
  S.2.a it is upwards exposed, or
  
  S.2.b it can be hoisted to its exit and is transparent in the block

- **Safety of hoisting out of the entry of a block**
  
  S.3 Hoist only if for each predecessor
  
  S.3.a it can be hoisted to its exit, or
  
  S.3.b it is available at its exit.
Revised Safety Criteria of Hoisting an Expression

- **Safety of hoisting to the exit of a block**
  
  S.1 Hoist only if it can be hoisted out of the entries of all successor blocks

- **Safety of hoisting to the entry of a block**
  
  S.2 Hoist only if
    
    S.2.a it is upwards exposed, or
    S.2.b it can be hoisted to its exit and is transparent in the block
  
  S.3 Hoist only if for each predecessor
    
    S.3.a it can be hoisted to its exit, or
    S.3.b it is available at its exit.
Desirability of Hoisting an Expression
Desirability of Hoisting an Expression

- Desirability of hoisting to the entry of a block
Desirability of Hoisting an Expression

- Desirability of hoisting to the entry of a block

**D.1** Hoist only if it is partially available
Final Hoisting Criteria

• **Safety of hoisting to the exit of a block**

  S.1 Hoist only if it can be hoisted out of the entries of all successor blocks

• **Safety of hoisting to the entry of a block**

  S.2 Hoist only if
  
  S.2.a it is upwards exposed, or
  S.2.b it can be hoisted to its exit and is transparent in the block

  S.3 Hoist only if for each predecessor
  
  S.3.a it can be hoisted to its exit, or
  S.3.b it is available at its exit.

• **Desirability of hoisting to the entry of a block**

  D.1 Hoist only if it is partially available
From Hoisting Criteria to Data Flow Equations (1)

First Level Global Data Flow Properties in PRE

- Partial Availability.

\[ PavIn_n = \begin{cases} BI & n \text{ is Start block} \\ \bigcup_{p \in \text{pred}(n)} PavOut_p & \text{otherwise} \end{cases} \]

\[ PavOut_n = \text{Gen}_n \cup (PavIn_n - \text{Kill}_n) \]

- Total Availability.

\[ AvIn_n = \begin{cases} BI & n \text{ is Start block} \\ \bigcap_{p \in \text{pred}(n)} AvOut_p & \text{otherwise} \end{cases} \]

\[ AvOut_n = \text{Gen}_n \cup (AvIn_n - \text{Kill}_n) \]
From Hoisting Criteria to Data Flow Equations (2)

- **Safety of hoisting to the exit of a block**
  
  **S.1** Hoist only if it can be hoisted out of the entries of all successor blocks

- **Safety of hoisting to the entry of a block**
  
  **S.2** Hoist only if
  
  **S.2.a** it is upwards exposed, or
  
  **S.2.b** it can be hoisted to its exit and is transparent in the block

  **S.3** Hoist only if for each predecessor
  
  **S.3.a** it can be hoisted to its exit, or
  
  **S.3.b** it is available at its exit.

- **Desirability of hoisting to the entry of a block**
  
  **D.1** Hoist only if it is partially available
From Hoisting Criteria to Data Flow Equations (2)

- **Safety of hoisting to the exit of a block**
  
  **S.1** Hoist only if it can be hoisted out of the entries of all successor blocks

- **Safety of hoisting to the entry of a block**
  
  **S.2** Hoist only if
  
  1. it is upwards exposed, or
  2. it can be hoisted to its exit and is transparent in the block

  **S.3** Hoist only if for each predecessor
  
  1. it can be hoisted to its exit, or
  2. it is available at its exit.

- **Desirability of hoisting to the entry of a block**

  **D.1** Hoist only if it is partially available
From Hoisting Criteria to Data Flow Equations (2)

- **Safety of hoisting to the exit of a block**
  
  S.1 Hoist only if it can be hoisted out of the entries of all successor blocks.

- **Safety of hoisting to the entry of a block**
  
  S.2 Hoist only if
  
  S.2.a it is upwards exposed, or
  
  S.2.b it can be hoisted to its exit and is transparent in the block.

  S.3 Hoist only if for each predecessor
  
  S.3.a it can be hoisted to its exit, or
  
  S.3.b it is available at its exit.

- **Desirability of hoisting to the entry of a block**
  
  D.1 Hoist only if it is partially available.

\[ \forall s \in \text{succ}(n), \quad \text{Out}_n \subseteq \text{In}_s \]
From Hoisting Criteria to Data Flow Equations (2)

- **Safety of hoisting to the exit of a block**
  
  S.1 Hoist only if it can be hoisted out of the entries of all successor blocks

- **Safety of hoisting to the entry of a block**
  
  S.2 Hoist only if
  
  S.2.a it is upwards exposed, or  
  S.2.b it can be hoisted to its exit and is transparent in the block

  S.3 Hoist only if for each predecessor
  
  S.3.a it can be hoisted to its exit, or  
  S.3.b it is available at its exit.

- **Desirability of hoisting to the entry of a block**
  
  D.1 Hoist only if it is partially available

\[ \forall s \in \text{succ}(n), \quad \text{Out}_n \subseteq \text{In}_s \]

\[ \text{In}_n \subseteq \text{AntGen}_n \cup (\text{Out}_n - \text{Kill}_n) \]
From Hoisting Criteria to Data Flow Equations (2)

- **Safety of hoisting to the exit of a block**
  
  S.1 Hoist only if it can be hoisted out of the entries of all successor blocks

- **Safety of hoisting to the entry of a block**
  
  S.2 Hoist only if
  - S.2.a it is upwards exposed, or
  - S.2.b it can be hoisted to its exit and is transparent in the block

  S.3 Hoist only if for each predecessor
  - S.3.a it can be hoisted to its exit, or
  - S.3.b it is available at its exit.

- **Desirability of hoisting to the entry of a block**
  
  D.1 Hoist only if it is partially available

\[ \forall s \in \text{succ}(n), \]
\[ \text{Out}_n \subseteq \text{In}_s \]

\[ \text{In}_n \subseteq \text{AntGen}_n \cup (\text{Out}_n - \text{Kill}_n) \]

\[ \forall p \in \text{pred}(n), \]
\[ \text{In}_n \subseteq \text{AvOut}_p \cup \text{Out}_p \]
From Hoisting Criteria to Data Flow Equations (2)

- **Safety of hoisting to the exit of a block**
  
  S.1 Hoist only if it can be hoisted out of the entries of all successor blocks

- **Safety of hoisting to the entry of a block**
  
  S.2 Hoist only if
  
  S.2.a it is upwards exposed, or
  
  S.2.b it can be hoisted to its exit and is transparent in the block

  S.3 Hoist only if for each predecessor
  
  S.3.a it can be hoisted to its exit, or
  
  S.3.b it is available at its exit.

- **Desirability of hoisting to the entry of a block**
  
  D.1 Hoist only if it is partially available

\[
\forall s \in \text{succ}(n), \quad \text{Out}_n \subseteq \text{In}_s
\]

\[
\text{In}_n \subseteq \text{AntGen}_n \cup (\text{Out}_n - \text{Kill}_n)
\]

\[
\forall p \in \text{pred}(n), \quad \text{In}_n \subseteq \text{AvOut}_p \cup \text{Out}_p
\]

\[
\text{In}_n \subseteq \text{PavlIn}_n
\]
From Hoisting Criteria to Data Flow Equations (2)

- **Safety of hoisting to the exit of a block**
  
  \( S.1 \) Hoist only if it can be hoisted out of the entries of all successor blocks

\[
\forall s \in \text{succ}(n), \quad \text{Out}_n \subseteq \text{In}_s
\]

- **Safety of hoisting to the entry of a block**
  
  \( S.2 \) Hoist only if
  
  \( S.2.a \) it is upwards exposed, or
  
  \( S.2.b \) it can be hoisted to its exit and is transparent in the block

\( S.3 \) Hoist only if for each predecessor

\( S.3.a \) it can be hoisted to its exit, or

\( S.3.b \) it is available at its exit.

- **Desirability of hoisting to the entry of a block**
  
  \( D.1 \) Hoist only if it is partially available

\[
\forall p \in \text{pred}(n), \quad \text{In}_n \subseteq \text{AvOut}_p \cup \text{Out}_p
\]

\[
\text{In}_n \subseteq \text{PavIn}_n
\]
∀s ∈ succ(n),
   \( Out_n \subseteq In_s \)

\( In_n \subseteq AntGen_n \cup (Out_n - Kill_n) \)

∀p ∈ pred(n),
   \( In_n \subseteq AvOut_p \cup Out_p \)

\( In_n \subseteq PavIn_n \)
From Hoisting Criteria to Data Flow Equations (3)

\[ \forall s \in \text{succ}(n), \quad \text{Out}_n \subseteq \text{In}_s \]

\[ \text{In}_n \subseteq \text{AntGen}_n \cup (\text{Out}_n - \text{Kill}_n) \]

\[ \forall p \in \text{pred}(n), \quad \text{In}_n \subseteq \text{AvOut}_p \cup \text{Out}_p \]

\[ \text{In}_n \subseteq \text{PavIn}_n \]

Find out the largest such set
∀s ∈ succ(n),
Out_n ⊆ In_s

In_n ⊆ AntGen_n ∪ (Out_n − Kill_n)

∀p ∈ pred(n),
In_n ⊆ AvOut_p ∪ Out_p

In_n ⊆ PavIn_n

Desirability: D.1

In_n = PavIn_n

Expressions should be partially available, and
From Hoisting Criteria to Data Flow Equations (3)

∀\(s \in \text{succ}(n)\),
\[\text{Out}_n \subseteq \text{In}_s\]

\[\text{In}_n \subseteq \text{AntGen}_n \cup (\text{Out}_n - \text{Kill}_n)\]

∀\(p \in \text{pred}(n)\),
\[\text{In}_n \subseteq \text{AvOut}_p \cup \text{Out}_p\]

\[\text{In}_n \subseteq \text{PavIn}_n\]

Safety: S.2.a

\[\text{In}_n = \text{PavIn}_n \cap (\text{AntGen}_n \cup \text{Out}_n - \text{Kill}_n)\]

Expressions should be upwards exposed, or
From Hoisting Criteria to Data Flow Equations (3)

\[ \forall s \in \text{succ}(n), \quad Out_n \subseteq In_s \]

\[ In_n \subseteq \text{AntGen}_n \cup (Out_n - \text{Kill}_n) \]

\[ \forall p \in \text{pred}(n), \quad In_n \subseteq \text{AvOut}_p \cup Out_p \]

\[ In_n \subseteq \text{PavIn}_n \]

\[ In_n = \text{PavIn}_n \cap (\text{AntGen}_n \cup (Out_n - \text{Kill}_n)) \]

Safety: S.2.b

Expressions can be hoisted to the exit and are transparent in the block

Jul 2017 IIT Bombay
From Hoisting Criteria to Data Flow Equations (3)

∀s ∈ succ(n),
Out_n ⊆ In_s

In_n ⊆ AntGen_n ∪ (Out_n − Kill_n)

∀p ∈ pred(n),
In_n ⊆ AvOut_p ∪ Out_p

In_n ⊆ PavIn_n

Safety: S.3.b

\[ In_n = \bigcap_{p \in \text{pred}(n)} (Out_p \cup \text{AvOut}_p) \]

\[ \bigcap \left( \text{AntGen}_n \cup (Out_n - Kill_n) \right) \]
From Hoisting Criteria to Data Flow Equations (3)

∀s ∈ succ(n),
Out_n ⊆ In_s

In_n ⊆ AntGen_n ∪ (Out_n − Kill_n)

∀p ∈ pred(n),
In_n ⊆ AvOut_p ∪ Out_p

In_n ⊆ PavIn_n

Safety: S.3.a

In_n = PavIn_n ∩ (AntGen_n ∪ (Out_n − Kill_n))

∩

Out_p ∪ AvOut_p

p ∈ pred(n)

...expressions are available at the exit of the same predecessor
∀ \( s \in \text{succ}(n) \),
\[ \text{Out}_n \subseteq \text{In}_s \]

\( \text{In}_n \subseteq \text{AntGen}_n \cup (\text{Out}_n - \text{Kill}_n) \)

∀ \( p \in \text{pred}(n) \),
\[ \text{In}_n \subseteq \text{AvOut}_p \cup \text{Out}_p \]

\( \text{In}_n \subseteq \text{PavIn}_n \)

\[
\text{In}_n = \text{PavIn}_n \cap \left( \text{AntGen}_n \cup (\text{Out}_n - \text{Kill}_n) \right) \cap \left( \bigcup_{p \in \text{pred}(n)} (\text{Out}_p \cup \text{AvOut}_p) \right)
\]

\[
\text{Out}_n = \begin{cases} 
\text{BI} & \text{if } n \text{ is End block} \\
\text{otherwise} & \end{cases}
\]

Boundary condition
From Hoisting Criteria to Data Flow Equations (3)

∀s ∈ succ(n),
Out_n ⊆ In_s

In_n ⊆ AntGen_n ∪ (Out_n − Kill_n)

∀p ∈ pred(n),
In_n ⊆ AvOut_p ∪ Out_p

In_n ⊆ PavIn_n

Safety: S.1

\[
In_n = PavIn_n \cap \left( \text{AntGen}_n \cup (Out_n - \text{Kill}_n) \right) \bigcap_{p \in \text{pred}(n)} \left( Out_p \cup \text{AvOut}_p \right)
\]

Out_n = \begin{cases} BI & n \text{ is End block} \\ \bigcap_{s \in \text{succ}(n)} In_s & \text{otherwise} \end{cases}

Expressions should be hoisted to the exit of a block if they can be hoisted to the entry of all successors.

Jul 2017
∀ s ∈ succ(n),
Out_n ⊆ In_s

ln_n ⊆ AntGen_n ∪ (Out_n − Kill_n)

∀ p ∈ pred(n),
In_n ⊆ AvOut_p ∪ Out_p

In_n ⊆ PavIn_n

\[
\begin{align*}
\text{In}_n &= \text{PavIn}_n \cap \left( \text{AntGen}_n \cup (\text{Out}_n - \text{Kill}_n) \right) \\
&\quad \cap \bigg( \text{Out}_p \cup \text{AvOut}_p \bigg) \\
\text{Out}_n &= \begin{cases} \text{BI} & \text{if } n \text { is End block} \\ \bigcap_{s \in \text{succ}(n)} \text{In}_s & \text{otherwise} \end{cases}
\end{align*}
\]
Anticipability and PRE (Hoistability) Data Flow Equations

<table>
<thead>
<tr>
<th>PRE Hoistability</th>
<th>Anticipability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ln_n = \text{PavIn}_n \cap (\text{AntGen}_n \cup (\text{Out}_n - \text{Kill}<em>n)) \cap \bigcup</em>{p \in \text{pred}(n)} (\text{Out}_p \cup \text{AvOut}_p)$</td>
<td>$\text{Out}<em>n = \begin{cases} \text{BL} &amp; n \text{ is End block} \ \bigcap</em>{s \in \text{succ}(n)} \ln_s &amp; \text{otherwise} \end{cases}$</td>
</tr>
</tbody>
</table>
Anticipability and PRE (Hoistability) Data Flow Equations

### PRE Hoistability

\[
In_n = \text{Pred}In_n \cap (\text{AntGen}_n \cup (\text{Out}_n - \text{Kill}_n)) \\
\bigcap_{p \in \text{pred}(n)} (\text{Out}_p \cup \text{AvOut}_p)
\]

\[
Out_n = \begin{cases} 
  \text{Bl} & \text{n is End block} \\
  \bigcap_{s \in \text{succ}(n)} In_s & \text{otherwise}
\end{cases}
\]

### Anticipability

\[
In_n = \text{AntGen}_n \cup (\text{Out}_n - \text{Kill}_n)
\]

\[
Out_n = \begin{cases} 
  \text{Bl} & \text{n is End block} \\
  \bigcap_{s \in \text{succ}(n)} In_s & \text{otherwise}
\end{cases}
\]
Anticipability and PRE (Hoistability) Data Flow Equations

**PRE Hoistability**

\[ In_n = P_{av}In_n \cap (\text{AntGen}_n \cup (\text{Out}_n - \text{Kill}_n)) \]

\[ \bigcap_{p \in \text{pred}(n)} (\text{Out}_p \cup \text{AvOut}_p) \]

\[ \text{Out}_n = \begin{cases} \bigcap_{s \in \text{succ}(n)} \text{In}_s & \text{n is End block} \\ \text{otherwise} \end{cases} \]

**Anticipability**

\[ In_n = \text{AntGen}_n \cup (\text{Out}_n - \text{Kill}_n) \]

\[ \text{Out}_n = \begin{cases} \bigcap_{s \in \text{succ}(n)} \text{In}_s & \text{n is End block} \\ \text{otherwise} \end{cases} \]
# Anticipability and PRE (Hoistability) Data Flow Equations

## PRE Hoistability

\[ \text{In}_n = \text{PavIn}_n \cap \left( \text{AntGen}_n \cup (\text{Out}_n - \text{Kill}_n) \right) \]

\[ \bigcap_{p \in \text{pred}(n)} \left( \text{Out}_p \cup \text{AvOut}_p \right) \]

\[ \text{Out}_n = \begin{cases} 
\text{BL} & \text{n is End block} \\
\bigcap_{s \in \text{succ}(n)} \text{In}_s & \text{otherwise}
\end{cases} \]

## Anticipability

\[ \text{In}_n = \text{AntGen}_n \cup (\text{Out}_n - \text{Kill}_n) \]

\[ \text{Out}_n = \begin{cases} 
\text{BL} & \text{n is End block} \\
\bigcap_{s \in \text{succ}(n)} \text{In}_s & \text{otherwise}
\end{cases} \]

---

**PRE Hoistability is anticipability restricted by**
Anticipability and PRE (Hoistability) Data Flow Equations

**PRE Hoistability**

\[
In_n = PavIn_n \cap (\text{AntGen}_n \cup (Out_n - Kill_n)) \bigcap_{p \in \text{pred}(n)} (Out_p \cup \text{AvOut}_p)
\]

\[
Out_n = \begin{cases} 
\text{BI} & \text{n is End block} \\
\bigcap_{s \in \text{succ}(n)} In_s & \text{otherwise}
\end{cases}
\]

**Anticipability**

\[
In_n = \text{AntGen}_n \cup (Out_n - Kill_n)
\]

\[
Out_n = \begin{cases} 
\text{BI} & \text{n is End block} \\
\bigcap_{s \in \text{succ}(n)} In_s & \text{otherwise}
\end{cases}
\]

**PRE Hoistability is anticipability restricted by**

- safety of hoisting and
- partial availability
Deletion Criteria in PRE

- An expression is redundant in node $n$ if
  - it can be placed at the entry (i.e. can be “hoisted” out) of $n$, AND
  - it is upwards exposed in node $n$.

\[
\text{Redundant}_n = \text{In}_n \cap \text{AntGen}_n
\]

- A hoisting path for an expression $e$ begins at $n$ if $e \in \text{Redundant}_n$
- This hoisting path extends against the control flow.
Insertion Criteria in PRE

• An expression is inserted at the exit of node $n$ is
  
  - it can be placed at the exit of $n$, AND
  - it is not available at the exit of $n$, AND
  - it cannot be hoisted out of $n$, OR it is modified in $n$.

  $$Insert_n = Out_n \cap (\neg AvOut_n) \cap (\neg In_n \cup Kill_n)$$

• A hoisting path for an expression $e$ ends at $n$ if $e \in Insert_n$
Performing PRE by Computing $In/Out$: Simple Cases (1)

1. $c = a \times b$

2. $d = a \times b$

$\Rightarrow$

1. $t = a \times b$

2. $c = t$

1. $t = b \times c$

2. $d = t$
Performing PRE by Computing *In*/Out: Simple Cases (1)

\[
\begin{align*}
1 \quad & c = a \times b \\
2 \quad & d = a \times b
\end{align*}
\]

\[
\Rightarrow \quad \begin{align*}
1 \quad & t = a \times b \\
2 \quad & d = t
\end{align*}
\]

<table>
<thead>
<tr>
<th>Node</th>
<th>First Level Values</th>
<th>Init.</th>
<th>Iter. 1</th>
<th>Iter. 2</th>
<th>Redund.</th>
<th>Insert</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AntGen</td>
<td>Kill</td>
<td>PavIn</td>
<td>AvOut</td>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Jul 2017
Performing PRE by Computing $In/Out$: Simple Cases (1)

1. $c = a \times b$
2. $d = a \times b$

$\Rightarrow$

1. $t = a \times b$
   $c = t$
2. $d = t$

<table>
<thead>
<tr>
<th>Node</th>
<th>First Level Values</th>
<th>Init.</th>
<th>Iter. 1</th>
<th>Iter. 2</th>
<th>Redund.</th>
<th>Insert</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$AntGen$</td>
<td>$Kill$</td>
<td>$PavIn$</td>
<td>$AvOut$</td>
<td>$Out$</td>
<td>$In$</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Performing PRE by Computing $\text{In}/\text{Out}$: Simple Cases (1)

1. $c = a \times b$

2. $d = a \times b$

$\Rightarrow$

1. $t = a \times b$

2. $d = t$

<table>
<thead>
<tr>
<th>Node</th>
<th>AntGen</th>
<th>Kill</th>
<th>PavIn</th>
<th>AvOut</th>
<th>Out</th>
<th>In</th>
<th>Out</th>
<th>In</th>
<th>Redund.</th>
<th>Insert</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Performing PRE by Computing \textit{In/Out}: Simple Cases (1)

\[ c = a \times b \]

\[ d = a \times b \]

\[ t = a \times b \]

\[ c = t \]

\[ d = t \]

<table>
<thead>
<tr>
<th>Node</th>
<th>First Level Values</th>
<th>Init.</th>
<th>Iter. 1</th>
<th>Iter. 2</th>
<th>Redund.</th>
<th>Insert</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AntGen</td>
<td>Kill</td>
<td>PavIn</td>
<td>AvOut</td>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Performing PRE by Computing \( \text{In}/\text{Out} \): Simple Cases (1)

\[
\begin{align*}
1 & : c = a \times b \\
2 & : d = a \times b
\end{align*}
\]

\[
\begin{align*}
1 & : t = a \times b \\
2 & : d = t
\end{align*}
\]

<table>
<thead>
<tr>
<th>Node</th>
<th>First Level Values</th>
<th>Init.</th>
<th>Iter. 1</th>
<th>Iter. 2</th>
<th>Redund.</th>
<th>Insert</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AntGen</td>
<td>Kill</td>
<td>PavIn</td>
<td>AvOut</td>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Performing PRE by Computing $In/Out$: Simple Cases (1)

Redundancy

1. $c = a \times b$
2. $d = a \times b$

No Insertion

1. $t = a \times b$
2. $d = t$

<table>
<thead>
<tr>
<th>Node</th>
<th>First Level Values</th>
<th>Init.</th>
<th>Iter. 1</th>
<th>Iter. 2</th>
<th>Redund.</th>
<th>Insert</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AntGen</td>
<td>Kill</td>
<td>PavIn</td>
<td>AvOut</td>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Performing PRE by Computing *In/Out*: Simple Cases (1)

Redundancy

No Insertion

\[ c = a \times b \]

\[ d = a \times b \]

\[ t = a \times b \]

\[ c = t \]

\[ d = t \]

This is an instance of Common Subexpression Elimination
Performing PRE by Computing $in/out$: Simple Cases (2)

\[ c = a \ast b \]
\[ a = 5 \]
\[ d = a \ast b \]

\[ \Rightarrow \]

\[ t = a \ast b \]
\[ c = t \]
\[ d = t \]
\[ a = 5 \]
\[ t = a \ast b \]
Performing PRE by Computing \( \text{In}/\text{Out} \): Simple Cases (2)

\[
\begin{align*}
2: & \quad c = a \times b \\
3: & \quad a = 5 \\
4: & \quad d = a \times b
\end{align*}
\]

\[
\begin{align*}
2: & \quad t = a \times b \\
3: & \quad c = t \\
4: & \quad d = t
\end{align*}
\]

<table>
<thead>
<tr>
<th>Node</th>
<th>First Level Values</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AntGen</td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Jul 2017
Performing PRE by Computing In/Out: Simple Cases (2)

\[ c = a \times b \]
\[ a = 5 \]
\[ d = a \times b \]

\[ t = a \times b \]
\[ c = t \]
\[ t = a \times b \]
\[ d = t \]

Node | First Level Values | Init. | Iter. 1 | Iter. 2 | Redund. | Insert
--- | --- | --- | --- | --- | --- | ---
| AntGen | Kill | PavIn | AvOut | Out | In | Out | In | Out | In |
| 4 | 1 | 0 | 1 | 1 | | | | |
| 3 | 0 | 1 | 0 | 0 | | | | |
| 2 | 1 | 0 | 0 | 1 | | | | |
| 1 | 0 | 0 | 0 | 0 | | | | |
Performing PRE by Computing \( In/Out \): Simple Cases (2)

\[
\begin{align*}
2: & \quad c = a \ast b \\
3: & \quad a = 5 \\
4: & \quad d = a \ast b
\end{align*}
\Rightarrow
\begin{align*}
2: & \quad t = a \ast b \\
3: & \quad a = 5 \\
4: & \quad d = t
\end{align*}
\]

<table>
<thead>
<tr>
<th>Node</th>
<th>First Level Values</th>
<th>Init.</th>
<th>Iter. 1</th>
<th>Iter. 2</th>
<th>Redund.</th>
<th>Insert</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( AntGen )</td>
<td>( Kill )</td>
<td>( PavIn )</td>
<td>( AvOut )</td>
<td>( Out )</td>
<td>( In )</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Performing PRE by Computing $In/Out$: Simple Cases (2)

\[
\begin{align*}
\text{Node} & \quad \text{First Level Values} & \quad \text{Init.} & \quad \text{Iter. 1} & \quad \text{Iter. 2} & \quad \text{Redund.} & \quad \text{Insert} \\
\text{AntGen} & \quad \text{Kill} & \quad \text{PavIn} & \quad \text{AvOut} & \quad \text{Out} & \quad \text{In} & \quad \text{Out} & \quad \text{In} & \quad \text{Out} & \quad \text{In} & \quad \text{Redund.} & \quad \text{Insert} \\
4 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & & & \\
3 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & & & \\
2 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & & & \\
1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & & & 
\end{align*}
\]
Performing PRE by Computing $In/Out$: Simple Cases (2)

\[ c = a \times b \]
\[ d = a \times b \]
\[ a = 5 \]
\[ t = a \times b \]

<table>
<thead>
<tr>
<th>Node</th>
<th>AntGen</th>
<th>Kill</th>
<th>PavIn</th>
<th>AvOut</th>
<th>Out</th>
<th>In</th>
<th>Out</th>
<th>In</th>
<th>Redund.</th>
<th>Insert</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Jul 2017
Performing PRE by Computing $In/Out$: Simple Cases (2)

Redundancy

Insertion

<table>
<thead>
<tr>
<th>Node</th>
<th>First Level Values</th>
<th>Init.</th>
<th>Iter. 1</th>
<th>Iter. 2</th>
<th>Redund.</th>
<th>Insert</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AntGen</td>
<td>Kill</td>
<td>PavIn</td>
<td>AvOut</td>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Jul 2017 IIT Bombay
Performing PRE by Computing $In/Out$: Simple Cases (3)

$$
\begin{align*}
1 & \text{ } \\
2 & a = b \times c \\
3 & \\
\Leftrightarrow & \\
1 & t = b \times c \\
2 & a = t \\
3 & 
\end{align*}
$$
Performing PRE by Computing $In/Out$: Simple Cases (3)

$$a = b \times c$$

$$t = b \times c$$

1. Node
2. First Level Values
3. Init.
4. Iter. 1
5. Iter. 2
6. Redund.
7. Insert

<table>
<thead>
<tr>
<th>Node</th>
<th>AntGen</th>
<th>Kill</th>
<th>PavIn</th>
<th>AvOut</th>
<th>Out</th>
<th>In</th>
<th>Out</th>
<th>In</th>
<th>Out</th>
<th>In</th>
<th>Redund.</th>
<th>Insert</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Performing PRE by Computing $In/Out$: Simple Cases (3)

$$a = b \times c$$

$$t = b \times c$$

<table>
<thead>
<tr>
<th>Node</th>
<th>First Level Values</th>
<th>Init.</th>
<th>Iter. 1</th>
<th>Iter. 2</th>
<th>Redund.</th>
<th>Insert</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AntGen</td>
<td>Kill</td>
<td>PavIn</td>
<td>AvOut</td>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Performing PRE by Computing $In/Out$: Simple Cases (3)

1. $a = b \times c$
2. $t = b \times c$
3. $a = t$

Node | First Level Values | Init. | Iter. 1 | Iter. 2 | Redund. | Insert
--- | --- | --- | --- | --- | --- | ---
AntGen | Kill | PavIn | AvOut | Out | In | Out | In | Out | In |
3 | 0 | 0 | 1 | 1 | 0 | 1 | |
2 | 1 | 0 | 1 | 1 | 1 | 1 | |
1 | 0 | 0 | 0 | 0 | 1 | 1 | |
Performing PRE by Computing $In/Out$: Simple Cases (3)

![Diagram showing the flow of operations and nodes]

<table>
<thead>
<tr>
<th>Node</th>
<th>First Level Values</th>
<th>Init.</th>
<th>Iter. 1</th>
<th>Iter. 2</th>
<th>Redund.</th>
<th>Insert</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AntGen</td>
<td>Kill</td>
<td>PavIn</td>
<td>AvOut</td>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Performing PRE by Computing In/Out: Simple Cases (3)

\[
\begin{align*}
1 &
\left\{ \begin{array}{l}
\text{Node} = 1 \\
\text{First Level Values} = 0
\end{array} \right.
\\
2 &
\left\{ \begin{array}{l}
\text{Node} = 2 \\
\text{First Level Values} = a = b \times c
\end{array} \right.
\\
3 &
\left\{ \begin{array}{l}
\text{Node} = 3 \\
\text{First Level Values} = 0
\end{array} \right.
\end{align*}
\]

\[
\begin{align*}
1 &
\left\{ \begin{array}{l}
\text{Node} = 1 \\
\text{First Level Values} = t = b \times c
\end{array} \right.
\\
2 &
\left\{ \begin{array}{l}
\text{Node} = 2 \\
\text{First Level Values} = a = t
\end{array} \right.
\\
3 &
\left\{ \begin{array}{l}
\text{Node} = 3 \\
\text{First Level Values} = 0
\end{array} \right.
\end{align*}
\]

\[
\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|}
\hline
\text{Node} & \text{AntGen} & \text{Kill} & \text{PavIn} & \text{AvOut} & \text{Init.} & \text{Iter. 1} & \text{Iter. 2} & \text{Redund.} & \text{Insert} \\
\hline
3 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
2 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\
\hline
\end{array}
\]
Performing PRE by Computing $\text{In/Out}$: Simple Cases (3)

There are three cases to consider, each represented by a node in the diagram:

1. **Insertion**:
   - 1st Node: $a = b \times c$
   - 2nd Node: $t = b \times c$
   - 3rd Node: $a = t$

2. **Redundancy**:
   - The redundancy is identified and marked.

The table below summarizes the first level values and the iterations for each node.

<table>
<thead>
<tr>
<th>Node</th>
<th>First Level Values</th>
<th>Init.</th>
<th>Iter. 1</th>
<th>Iter. 2</th>
<th>Redund.</th>
<th>Insert</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AntGen</td>
<td>Kill</td>
<td>PavIn</td>
<td>AvOut</td>
<td>Out</td>
<td>In</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The table shows the initial and iterated values for each node's input and output, indicating the presence or absence of redundancy and whether an insertion is required.
Tutorial Problems for PRE

(a)

1

2

\(a \ast b\)

3

4

Jul 2017
Tutorial Problems for PRE

(a)
Tutorial Problems for PRE

(a)

(b)
Tutorial Problems for PRE

(a) $a \times b$

(b) $a = 5$

Jul 2017
Tutorial Problems for PRE

(a) $a \times b$

(b) $a = 5$

(c) $a \times b$

$I = 0$

$O = 1$

$1$ to $2$

$2$ to $3$

$3$ to $4$

Jul 2017

IIT Bombay
Tutorial Problems for PRE

(a) 

(b) 

(c)
Tutorial Problems for PRE

(a) $a \times b$
(b) $a = 5$
(c) $a \times b$
(d) $a \times b$
Tutorial Problems for PRE

(a) \(a \times b\)

(b) \(a = 5\)

(c) \(a \times b\), \(a = 5\)

(d) \(a \times b\), \(a = 5\)
Tutorial Problems for PRE

(a)  

(b)  

(c)  

(d)  

(e)  

\( a \times b \)

\( a = 5 \)

\( a \times b \)

\( a = 5 \)

\( a \times b \)

\( a = 5 \)

\( a \times b \)

\( a = 5 \)

\( a \times b \)
Tutorial Problems for PRE

(a) $a \times b$

(b) $a = 5$

(c) $a \times b$, $a = 5$

(d) $a \times b$, $a = 5$

(e) $a \times b$, $a = 5$
Tutorial Problems for PRE

(a) $a * b$

(b) $a = 5$

(c) $a * b$

(d) $a * b$

(e) $a * b$

Redundancy

Insertion
Further Tutorial Problem for PRE

Let \( \{a \ast b, b \ast c\} \equiv \text{bit string 11} \)

<table>
<thead>
<tr>
<th>Node ( n )</th>
<th>( \text{Kill}_n )</th>
<th>( \text{AntGen}_n )</th>
<th>( \text{PavIn}_n )</th>
<th>( \text{AvOut}_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>2</td>
<td>00</td>
<td>10</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>00</td>
<td>11</td>
<td>00</td>
</tr>
<tr>
<td>4</td>
<td>00</td>
<td>00</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>00</td>
<td>01</td>
<td>11</td>
<td>01</td>
</tr>
<tr>
<td>6</td>
<td>00</td>
<td>00</td>
<td>11</td>
<td>01</td>
</tr>
</tbody>
</table>

- Compute \( \text{In}_n/\text{Out}_n/\text{Redundant}_n/\text{Insert}_n \)
- Identify hoisting paths
### Result of PRE Data Flow Analysis of the Running Example

Bit vector: \( a \cdot b \ | \ a + b \ | \ a - b \ | \ a - c \ | \ b + c \)

<table>
<thead>
<tr>
<th>Block</th>
<th>Constant information</th>
<th>Global Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Iteration # 1</td>
</tr>
<tr>
<td></td>
<td>PavIn_( n )</td>
<td>AvOut_( n )</td>
</tr>
<tr>
<td>( n_8 )</td>
<td>11111</td>
<td>00011</td>
</tr>
<tr>
<td>( n_7 )</td>
<td>11101</td>
<td>11000</td>
</tr>
<tr>
<td>( n_6 )</td>
<td>11101</td>
<td>11001</td>
</tr>
<tr>
<td>( n_5 )</td>
<td>11101</td>
<td>11000</td>
</tr>
<tr>
<td>( n_4 )</td>
<td>11100</td>
<td>10100</td>
</tr>
<tr>
<td>( n_3 )</td>
<td>11101</td>
<td>10000</td>
</tr>
<tr>
<td>( n_2 )</td>
<td>10001</td>
<td>00010</td>
</tr>
<tr>
<td>( n_1 )</td>
<td>00000</td>
<td>10001</td>
</tr>
</tbody>
</table>

Jul 2017

IIT Bombay
Hoisting Paths for Some Expressions in the Running Example

\begin{align*}
&\text{n}_1: \quad b = 4; \\
&\quad a = b + c; \\
&\quad d = a \ast b; \\
&\text{n}_2: \quad b = a - c; \\
&\text{n}_3: \quad c = b + c; \\
&\text{n}_4: \quad c = a \ast b; \\
&\quad f(a - b); \\
&\text{n}_5: \quad d = a + b; \\
&\text{n}_6: \quad f(b + c); \\
&\text{n}_7: \quad g(a + b); \\
&\text{n}_8: \quad h(a - c); \\
&\quad f(b + c);
\end{align*}
**Hoisting Paths for Some Expressions in the Running Example**

- $n_1$: $b = 4;\newline a = b + c;\newline d = a \times b;$
- $n_2$: $b = a - c;$
- $n_3$: $c = b + c;$
- $n_4$: $c = a \times b;\newline f(a - b);$  
  - $n_5$: $d = a + b;$
  - $n_6$: $f(b + c);$  
  - $n_7$: $g(a + b);$  
  - $n_8$: $h(a - c);\newline f(b + c);$
Hoisting Paths for Some Expressions in the Running Example

\[ n_1 \]
\[ b = 4; \]
\[ a = b + c; \]
\[ d = a \times b; \]

\[ n_2 \]
\[ b = a - c; \]

\[ n_3 \]
\[ c = b + c; \]

\[ n_4 \]
\[ c = a \times b; \]
\[ f(a - b); \]

\[ n_5 \]
\[ d = a + b; \]

\[ n_6 \]
\[ f(b + c); \]

\[ n_7 \]
\[ g(a + b); \]

\[ n_8 \]
\[ h(a - c); \]
\[ f(b + c); \]
Hoisting Paths for Some Expressions in the Running Example

\begin{align*}
\text{n1:} & \quad b = 4; \\
& \quad a = b + c; \\
& \quad d = a \times b; \\
\text{n2:} & \quad b = a - c; \\
\text{n3:} & \quad c = b + c; \\
\text{n4:} & \quad c = a \times b; \\
& \quad f(a - b); \\
\text{n5:} & \quad d = a + b; \\
\text{n6:} & \quad f(b + c); \\
\text{n7:} & \quad g(a + b); \\
\text{n8:} & \quad h(a - c); \\
& \quad f(b + c); \\
\end{align*}
Hoisting Paths for Some Expressions in the Running Example

\[ b = 4; \]
\[ a = b + c; \]
\[ d = a \ast b; \]

\[ n_1 \]

\[ b = a - c; \]

\[ n_2 \]

\[ c = b + c; \]

\[ n_3 \]

\[ c = a \ast b; \]

\[ f(a - b); \]

\[ n_4 \]

\[ d = a + b; \]

\[ n_5 \]

\[ f(b + c); \]

\[ n_6 \]

\[ g(a + b); \]

\[ n_7 \]

\[ h(a - c); \]

\[ f(b + c); \]

\[ n_8 \]
Hoisting Paths for Some Expressions in the Running Example

\[ b = 4; \]
\[ a = b + c; \]
\[ d = a \times b; \]
\[ c = b + c; \]
\[ d = a + b; \]
\[ f(a - b); \]
\[ f(b + c); \]
\[ g(a + b); \]
\[ h(a - c); \]
\[ f(b + c); \]
Optimized Version of the Running Example

\[ b = 4; \]
\[ t_2 = b + c; \]
\[ a = t_2; \]
\[ t_0 = a \times b; \]
\[ d = t_0; \]

\[ c = t_2 \]
\[ t_1 = a + b; \]

\[ b = c; \]
\[ f(a - c); \]
\[ t_2 = b + c; \]

\[ c = t_0; \]
\[ f(a - b); \]
\[ t_2 = b + c; \]

\[ d = t_1; \]
\[ t_2 = b + c; \]

\[ f(t_2); \]

\[ g(t_1); \]

\[ h(a - c); \]
\[ f(t_2); \]